127 MAY 1987

2.95 (3.95 CANADA)

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

Notes On Computer Music

Getting Started with MIDI

Scientific Programming with Dimensional Data Types

Ray Duncan: Command Processors

Languages:
Ada
C Subroutines
True BASIC and Modula-2
Object-Oriented LISP
New BASIC Subroutines

The most up-to-date training in the UNIX® System, from the people who keep the UNIX System up-to-date.

What could make more sense than UNIX System training from the people who invented UNIX —the people responsible for all its updates and revisions. AT&T.

Created to train AT&T's own professionals, our curriculum is the most comprehensive available—including C language as well as UNIX. And every course is practical and job-related.

Training for everyone

☐ Systems developers
☐ Applications programmers
☐ Technical specialists
☐ System managers and users

Whatever your specialty, AT&T has the right curriculum, from basic overviews to programming to business applications and data communications. And every course is kept up-to-the-minute with such recent advances as System V Release 3.0.

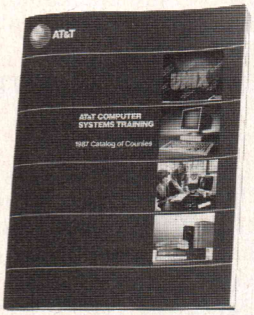
Individual attention

Classes are limited in size, so that each student can be given individual instruction and supervision. In laboratory classes, each student is assigned his own terminal. Instruction is by AT&T UNIX veterans and is personal, thorough, productive.

Classes forming now

Reserve as quickly as possible for preferred dates at our completely equipped training centers in Atlanta, Chicago, Dublin, OH, Los Angeles, Princeton, NJ, and Sunnyvale, CA. Or we'll arrange instruction on your site at your convenience. But don't wait—call or write now for information and seat reservations.

Free fact-packed training catalog:
Call 1 800 247-1212 or mail this coupon.



Registrar, AT&T Training,
P.O. Box 45038, Jacksonville, FL 32232-9974

Please rush me your course catalog with information on:
☐ UNIX System training ☐ UNIX System video training
☐ Data communications and networking training

Name ______
Title ______
Company ______
Address ______
City ______ State ______ Zip ______
Phone ( )

AT&T
The right choice.

CONTENTS

ARTICLES

Computers and music ▶

MUSIC: Pushing the Sound Envelope 16
by David Levitt
A brief history of computer music and a look at some recent developments in MIDI programming, sampling, transient-oriented synthesis methods, and programs that compose and collaborate on original music.

Getting started with MIDI ▶

MUSIC: Designing a Music Recorder 22
by Mark Garvin
Mark shows how to design a software-based music recorder using MIDI.

Dimensional data types in Ada ▶

SCIENTIFIC: Dimensional Data Types 50
by Do-While Jones
Using dimensional units as data types can facilitate the writing of clearer, more easily maintained code. Do-While presents example programs in Ada.

COLUMNS

C filters in statistics ▶

C CHEST 102
by Allen Holub
Allen looks at statistical applications of digital low-pass filters, a set of subroutines that has applications in both scientific and music programming.

Command processors ▶

16-BIT SOFTWARE TOOLBOX 118
by Ray Duncan
Ray looks at some "quality of life tools" for MS-DOS programmers, namely, Command Plus, and ProCED.

True BASIC and Modula-2 ▶

STRUCTURED PROGRAMMING 124
by Namir Clement Shammas
Namir analyses True BASIC modules and compares modules of True BASIC and Modula-2.

Object-oriented LISP ▶

ARTIFICIAL INTELLIGENCE 132
by Ernest R. Tello
Ernie discusses the features of PC Scheme—"the Turbo Pascal of object-oriented LISPs."

FORUM

Ignoble and mercenary motives ▶

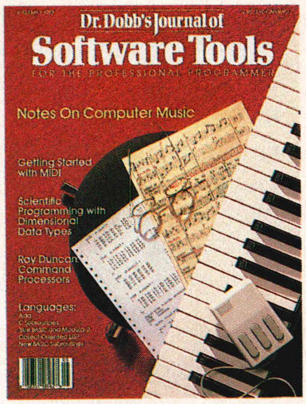
EDITORIAL 6
by Michael Swaine
RUNNING LIGHT 8
by Allen Holub
ARCHIVES 8
LETTERS 10
by you

Things computers can never do ▶

VIEWPOINT 14
by Philip J. Erdelsky
SWAINE'S FLAMES 152
by Michael Swaine

PROGRAMMER'S SERVICES

THE STATE OF BASIC: 144
More on new BASIC subroutines
BOOKS: 146
Numerical Recipes: The Art of Scientific Computing
OF INTEREST: 148
New products out there
ADVERTISER INDEX: 151
Where to find those ads



About the Cover

Programmers who have always longed to play (or play with) music and musicians with techie tendencies and a yen to create new sounds would feel right at home in the tableau pictured on this month's cover.

This Issue

What is it that draws so many programmers towards making music? Maybe it's the interactiveness of the process or the fact that a piece of music is something that can be shared with everyone (unlike a good piece of code, which only other programmers can appreciate), or maybe it's because computer music is a new frontier. Most likely it is a combination of these factors, plus the fact that making tools for making music is as fun and interesting as the end result. Here we look at the roots of computer music and at some interesting work to be done in this area on today's microcomputers. We also look at scientific programming with an article on dimensional data types by Do-While Jones, and Allen Holub covers both music and scientific programming in C Chest.

Next Issue

We've planned some very practical pieces for June. The lead article presents an algorithmic solution to the problem of sharing on-line resources using large priority queuing. We'll also have how-to articles about an extended communications port driver and building a two-bit analog to digital convertor.

The fastest C

Your search for execution speed is over. The new Microsoft® C Compiler Version 4.0 is here. With blazing performance. We've added common sub-expression elimination to our optimizer that produces code that rips through the benchmarks faster than ever before.

"...the Microsoft performance in the benchmarks for program execution is the best of the lot overall." —William Hunt, *PC Tech Journal*, January, 1986.*

But speed isn't the only edge you get with Microsoft C. Other advantages include a variety of memory models like our new HUGE model that breaks the 64K limit on single data items. Plus our NEAR, FAR and HUGE pointers, which provide you greater flexibility. All this allows you to fine tune your program to be as small and fast as possible.

"Excellent execution times, the fastest register sieve, and the best documentation in this review ...Microsoft Corporation has produced a tremendously useful compiler." —Christopher Skelly, *Computer Language*, February, 1986.

No more debugging hassles. Introducing CodeView. Free.

Now, for a limited time, we'll give you an unprecedented programming tool when you buy Microsoft C, free. New Microsoft CodeView™ offers the most powerful tool yet in the war on C bugs. Forget the hex dumps. Now you can view and work with programs at any level you want. Use the program source, the disassembled object code, or

Microsoft C Compiler Version 4.00

Microsoft C Compiler

- Produces fast executables and optimized code including elimination of common sub-expressions. NEW!
- Implements register variables.
- Small, Medium and Large Memory model libraries.
- Compact and HUGE memory model libraries. NEW!
- Can mix models with NEAR, FAR and the new HUGE pointers.
- Transport source and object code between MS-DOS® and XENIX® operating systems.
- Library routines implement most of UNIX™ System V C library.
- Start-up source code to help create ROMable code. NEW!
- Full proposed ANSI C library support (except clock). NEW!
- Large number of third party support libraries available.
- Choose from three math libraries and generate in-line 8087/80287 instructions or floating point calls:
  - floating point emulator (utilizes 8087/80287 if installed).
  - 8087/80287 coprocessor support.
  - alternate math package – extra speed without an 8087/80287.
- Link your C routines with Microsoft FORTRAN (version 3.3 or higher), Microsoft Pascal (version 3.3 or higher) or Microsoft Macro Assembler.
- Microsoft Windows support and MS-DOS 3.1 networking support.
- Supports MS-DOS pathnames and input/output redirection.

Microsoft Program Maintenance Utility. NEW!

- Rebuilds your applications after your source files have changed.
- Supports macro definitions and inference rules.

Other Utilities

- Library Manager.
- Object Code Linker.
- EXE File Compression Utility.
- EXE File Header Utility.

C Benchmarks

In seconds

	Microsoft C 4.0	Lattice C 3.0	Computer Innovation C 2.3	Aztec C86 3.2	Wizard C 3.0
Sieve of Eratosthenes (register)	82.9	151.4	172.3	88.0	91.9
Copy Block	86.9	231.7	199.0	123.8	189.5

Run on an IBM PC XT with 512K memory

Microsoft CodeView
Window-oriented source-level debugger. NEW!

- Watch the values of your local and global variables and expressions as you debug.
- Set conditional breakpoints on variables, expressions or memory; trace and single step.
- Watch CPU registers and flags as you execute.
- Effectively uses up to four windows.
- Debug using your original source code, the resulting disassembly or both intermingled.
- Use drop-down menus to execute CodeView commands.
- Access the on-line help to lead you through CodeView's options and settings.
- Easily debug graphics-oriented programs since program output is kept separate from debugger output.
- Keyboard or optional mouse support.
- Enter in familiar SYMDEB or DEBUG commands.

*Reprinted from *PC Tech Journal*, January 1986, copyright 1986, Ziff-Davis Publishing.

you've ever seen.

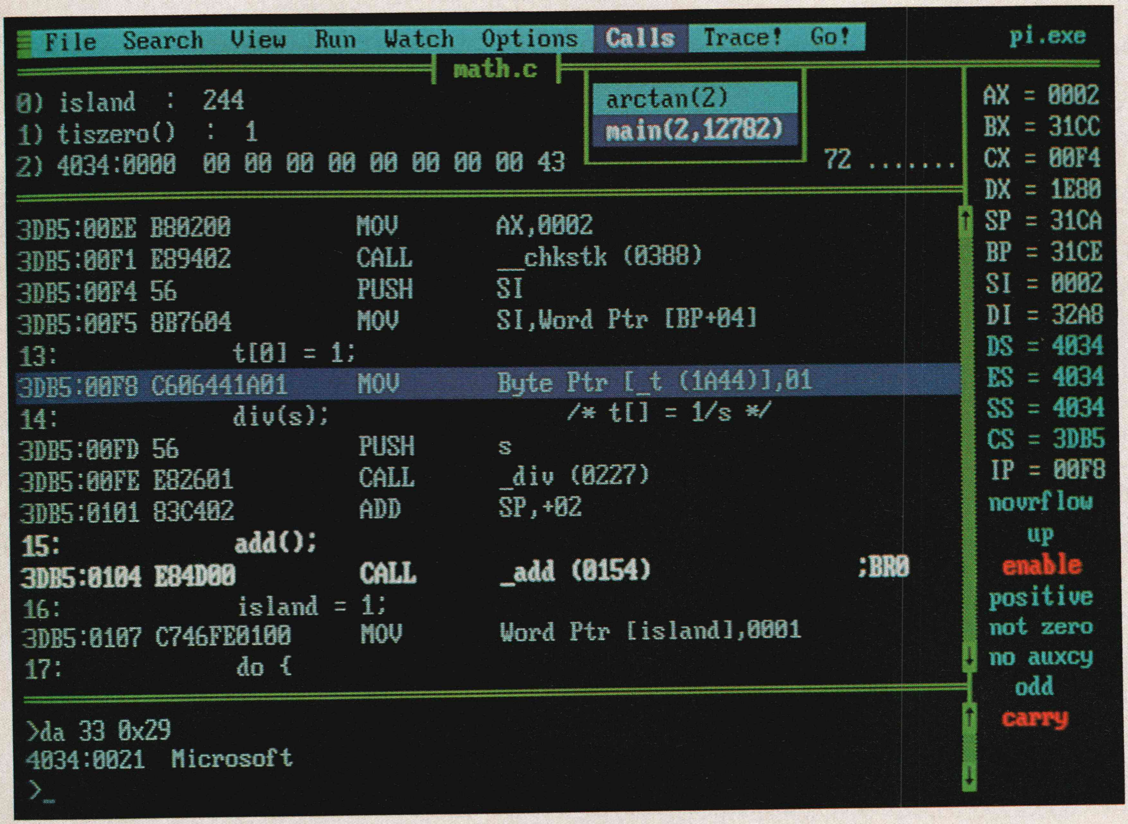
both at the same time. Open a window to view CPU registers and flags. Watch local and global variables as well. All while your program is running.

CodeView gives you complete control. Trace execution a line at a time—using source or assembly code. Or set conditional breakpoints on variables, memory or expressions. CodeView supports the familiar SYMDEB command syntax, as you'd expect. Commands are also available through drop-down menus. Combine the new window-oriented interface with our on-line help and debugging has never been easier. Or quicker.

Take the $5 CodeView tour.

You may find it hard to believe our debugger can do all we've claimed. So we're offering test drives. Five bucks will put you behind the wheel of a Microsoft C demo disk with CodeView.† See for yourself how fast debugging can get.

For more information about the CodeView demo disk, the new Microsoft C Compiler, a list of third party library support or the name of your nearest Microsoft dealer, call (800) 426-9400. In Washington State and Alaska, (206) 882-8088. In Canada call (416) 673-7638.

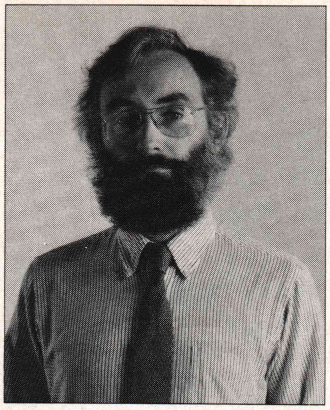


Microsoft® C Compiler

The High Performance Software

CIRCLE 380 ON READER SERVICE CARD

EDITORIAL



When giants sneeze, they set the sod ashiver and shake the sedentary stones.

When the Lotus Development corporate nose got irritated by products that copied the look and feel of its successful spreadsheet product, it blew up a lot of dirt, and from under scattered rocks slithered and scuttered the hungry lawyers.

At press time, things were looking bad for the lawyers.

Lotus had brought suit against Paperback Software and Mosaic Software over alleged copyright infringement of Lotus 1-2-3 by these companies' products, VP-Planner and The Twin, respectively. Lotus was not contending that these companies copied Lotus code but rather that their products, in being keystroke-for-keystroke compatible with 1-2-3, infringed on the copyrighted "look and feel" of 1-2-3.

Now that the copyright office has determined that there is no copyrightable look and feel to 1-2-3, Lotus' case is much weaker, but the issues the case raised will not go away. Critics such as Dan Bricklin, who created VisiCalc, the first electronic spreadsheet, have argued that to protect the keystoke sequences of a product denies competitors access to perhaps the most compelling feature a commercial software product can have: familiarity.

The suit, or one like it, could have enormous implications for innovation and advance in software development. Although VP-Planner and The Twin are anything but innovative, a decision against the imitators in such a case could make even truly innovative developers more cautious in bringing products to market. It also would remove one incentive to improve existing products. As Bricklin put it, "you won't have to do version two, because nobody else will be able to."

At least one editor has questioned the motives of Lotus executives, but surely their motives are beyond question. In this particular instance, the motives of the executives at Lotus and Paperback and Mosaic were ignoble and mercenary. Innovation issues have been raised by the Lotus look-and-feel case, to be sure, but the actions and motivations of the corporations involved had to do not with innovation but with profits.

These executives were undoubtedly behaving appropriately in this. Corporations exist to make profits for their stockholders, not to innovate, pioneer, or upgrade products, except as such actions may seem to them to be necessary steps to profits. These corporate executives smelled wealth in the well-trod ground of the Lotus 1-2-3 user interface. If the companies involved in the Lotus suit did not take a particularly high-minded view, it was because they were keeping their eyes on the turf.

As a market, the 1-2-3 turf is rich indeed, but, surveying it technologically, the ground these companies chose to squabble over is played out. The pace of technological development in personal computer software has left Lotus 1-2-3 behind, even though it is still healthy as a product.

But what-can-be still can drive what-will-be in the software market, and before long someone will bring the spreadsheet market up to date with the technology. I look forward to some innovative competitor leaving Lotus 1-2-3 and its archaic interface in the dust—even if it has to be Microsoft.

Michael Swaine

Michael Swaine
editor-in-chief

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly by M&T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points.

Article Submissions: Send manuscripts and disk (with article and listings) to the Editor.

***DDJ* on CompuServe:** Type GO DDJ

Address Correction Requested: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 0888-3076**

Customer Service: For subscription problems call: outside CA (800) 321-3333; in CA (619) 485-9623 or 566-6947. For book/software order problems call (415) 366-3600.

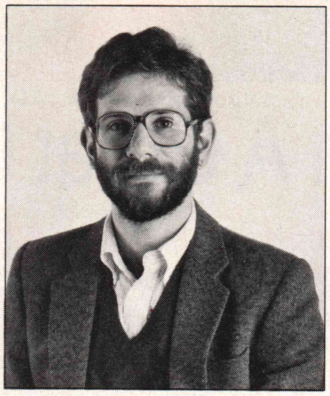
Subscriptions: $29.97 per 1 year; $56.97 for 2 years. Canada and Mexico add $27 per year airmail or $10 per year surface. All other countries add $27 per year airmail. Foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX: 620430 (WUI).

People's Computer Company

Dr. Dobb's Journal of Software Tools is published by M&T Publishing Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

RUNNING LIGHT



Most of the content of this issue has to do with either scientific programming or music. Scientific programming yes, but why music in a programmers' magazine?

Maybe you think the connection needs no explanation, particularly if you, like many professional programmers, are a musician. It seems that just as visual artists often support themselves doing paste-up, many musicians earn a living as programmers. In any case, there is a remarkably large overlap between computer programmers and musicians.

But more to the point, the application of computers to musical problems is often fascinating from a pure programming perspective. A truly interactive music system has to understand what's happening musically—a nontrivial problem in artificial intelligence. Moreover, music programs have some of the most complex, and most interesting, user interfaces going. They use graphics to a higher degree than most programs do and have to do sophisticated things on the output side, juggling the control of several other computers and synthesizers. What is MIDI if not a multiprocessor system communicating over a network?

What I'm saying here is that the problems of computer music are most often programming problems, of interest to the musician and nonmusician alike.

This issue covers the music theme with three music-related articles. David Levitt's piece discusses some of the newer trends in music-related hardware and software. It may pique the curiosity of even tone-deaf coders enough to get them interested in solving some of the programming problems David presents. Mark Garvin discusses MIDI, but from the not-often-taken perspective of how to store and retrieve the staggering amount of data that comes across the network. Finally, this month's C Chest discusses digital low-pass filters and demonstrates the universality of music software by applying these filters to statistical applications.

The one sour note in this issue, to my ear at least, is that there's not a whole lot of code. I'm hoping that those of you who are both programmers and musicians will send us articles about some of the programs you've written. If you have something useful you'd like to share, send it in.

Maybe together we can advance the state of the music software art.

Allen Holub
technical editor

ARCHIVES

Math in *DDJ*

"My experiences during the the last few months vividly illustrate the fact that there are plenty of good mathematical problems still waiting to be solved almost everywhere you look—especially in areas of life where mathematics has rarely been applied before. Mathematicians can provide solutions to these problems, receiving a double payoff—namely the pleasure of working out the mathematics, together with the appreciation of the people who can use the solutions. So let's go forth and apply mathematics in new ways."—*"Mathematical Typography," Donald E. Knuth, DDJ, March 1980.*

"I use a Polymorphic 8813 as a home system. As a precision buff, I was delighted by their newest release of BASIC which among the many other features, has a variable precision 'settable' from 6 to 26 digits. By the way, this precision holds for all of the trigonometric and other math functions unlike so-called double precision calculations in some other BASICs."—*Letter to the Editor, John W. McGraw, DDJ, March 1980.*

"I trust that anyone with even the slightest love for mathematics (however deep) will want to see how mathematical tools can be applied to the problems of programming."—*"An Introduction to Algorithm Design," Jon Louis Bentley, DDJ, April 1980.*

Ten Years Ago in *DDJ*

"Computers are considered to be useful tools with which to achieve a specific end result such as processing a payroll or calculating a trajectory. This view of computers has often been carried over into educational applications with the computer cast in the role of teacher/tutor. The low-cost home/school system described here (called FRED—Flexible Recreational and Educational Device) is intended as a plaything which encourages experimentation and stimulates a desire to learn.—*"A Practical, Low-Cost, Home/School Microprocessor System," Joe Weisbecker, DDJ, May 1977.*

"One difference from other versions of [computer game] CHASE—if two robots collide in my version, they do not annhialate [sic], but travel as a pair—with some strangely unpredictable consequences! (This was originally a 'bug' in my program, but it was so cute I decided to leave it in!)"—*"Video Chase for 8080/VDM," Joseph Jay Sanger, DDJ, May 1977.*

Dr. Dobb's Journal of
COMPUTER
Calisthenics & Orthodontia
Running Light Without Overbyte

LETTERS

BASIC09 and OS-9

Dear *DDJ*,

Brian Capouch was mistaken when he asserted in "The OS-9 Operating System" (January 1987) that the language BASIC09 and the operating system OS-9 "appeared together in 1981, a few months after the 6809 came into production." For example, the May 1979 issue of *Byte* contained an ad from Southwest Technical Products offering both an MP-09 Processor Card and a 68/09 Computer w/ 48K. As I recall, these products were being shipped in the second half of 1979.

This confusion is especially misleading because the reason for the BASIC09 project (which came to include OS-9) was to promote the 6809 processor: BASIC09 was to have provided an example of efficient coding, fast execution, and also a language-on-a-ROM for use by OEMs. I was the supervising engineer for the project, and it was especially unfortunate for me that it had still failed to terminate by early 1981, two years after the 6809 was introduced. The 8-bit 6809 was competing with the 16-bit 8088 and often came up "a day late and a dollar short" by paper comparison. BASIC09 would have helped, but by 1981 the new-design window for the 6809 had pretty well closed.

BASIC09 was to be composed of dynamically replaceable memory modules; OS-9 was to be multitasking, real-time, reentrant, and dynamically reconfigurable. Of course, these are great buzzwords now, but consider how different these concepts are from the current MS-DOS system and how much better off we'd be if MS-DOS had been forced, by competition, to include these features.

Terry Ritter
2609 Choctaw Trail
Austin, TX 78745

Hashing

Dear *DDJ*,

In response to Edwin T. Floyd's "Hashing for High-Performance Searching" (February 1987), I present the following items:

1. A hash code "qualifier" can be stored (along with the physical address of the data) in the hash bucket. The qualifier can be derived by a function similar to the hash function. When the hash bucket is searched for the desired key, the qualifier will most likely be unique for each different key. A physical read of the actual data item is then required to ensure a correct match; however, a hash miss becomes far less likely using this method. Because the data is kept separate from the index, it has to be read anyway (especially in the case of a duplicate key), so this method involves extra overhead only in the case of a hash miss or because of the presence of a duplicate key. The major advantages of this method are that the index space required is greatly reduced and the search time for lengthy keys is minimized. For 32-byte keys, for example, a linear index (for binary search) would require at least 34 bytes per entry (32 for the key, 2 for the data address). Assuming a 4-byte hash code qualifier, however, an index entry in a hash bucket would occupy a mere 6 bytes, with the search taking correspondingly less time.
2. Hashing techniques used as a disk database indexing scheme are not much different from the RAM-resident symbol table implementation presented by Floyd. Depending upon the size of the hash table, all or part of the table can be kept in memory, with some kind of buffering technique being an integral part.

A quick-access, hash-indexed database using the technique described above is available from us. It is a disk database offering hash-keyed and associative (relational) access simultaneously, which hints at the extreme versatility of hash indexing techniques. Although it does not make use of the move-to-front technique, there is no reason why this optimization could not be used. It seems to me that MTF optimization would be especially useful when duplicate keys are involved.

Dave Joy
Joy Research &
Development
9403 Wallingham Dr.
Spring, TX 77379

Dear *DDJ*,

I was pleased to see hashing discussed in two places in the February issue. It is my experience in developing practical commercial pro-

"I'm the same soulless, talentless hack as always, but MIDI makes the job a lot easier."

Building a Custom Operating System

With Operating System Toolbox you can design your own custom operating system.

Last month, we gave an introduction to the features of Wendin's Operating System Toolbox, a software construction set for IBM PC's and compatibles. This month we'll describe how to write a shell and link it with the Toolbox to create a custom operating system. If you haven't got one already, pick up a copy of the Toolbox from Wendin and follow along. The source code to the examples in this column is available on our own bulletin board at (509) 624-8093, 1200 baud.

The Shell

The shell is one of the most important parts of the operating system, since it determines how the system will interact with the user. The shell is simply a function written in C. Whenever a new process is created, it starts executing the shell. The shell can examine the status of the current process by referencing fields in the *process control block*, an internal data structure maintained by the Toolbox kernel. Such fields include the *process status longword* (PSL), its priority and its state.

Our example allows two kinds of processes: *user processes*, which accept commands from a terminal and execute them; and *subprocesses*, which just execute one program and quit. The shell can distinguish between them by examining the PSL_SUBPROC bit in the current process' PSL. A user command can create a process with this bit set to run a single program, or with it clear, to create another user process. This is as easy as giving a different number in one parameter of a create process system call, CREPRC.

Shell Structure

A user process basically performs two tasks over and over again: read a command from the user, and process that command. This is done until a command to exit the operating system is given. Input is read from the user with a QIO or RMS system call on the standard input device. A batch file will just have a filename instead of the name of a terminal here. Optionally, the shell can allow editing of input with function keys and arrow keys; this requires that the input function differentiate between file and non-file input devices.

Once the input is read, it is ready to be processed. For simple shells, an *if...else if...else* structure with string comparisons will suffice. For larger shells with more commands, it is easier to build a table of strings and an accompanying table of pointers to functions, so the shell can scan through the table and execute the corresponding function when it finds a match.

A very important command to include in your shell is the command to run a program. You can run a program in your own process with an EXEIMG call, or as a separate process by calling CREPRC. In our example, CREPRC is used to illustrate the use of PSL_SUBPROC to differentiate between processes. When CREPRC is called, you give it the name of a program to execute, the names of its standard input, output, and error devices, and several other parameters. The new copy of the shell that is executed when the new process starts is responsible for opening the standard devices and loading the program.

If you want the user process to wait until the program finishes before reading another command, you have to wait for the process to delete itself. When this occurs, an event flag will be set for the current process. To wait for this to happen, just use the WAITFR system service. If you want the program to run concurrently, just leave off the wait, and your shell will continue as soon as the other process has been created.

The Example Shell

Our example shell contains two internal commands and the ability to run programs. The PS command gives a list of the processes currently in the system. The EXIT command calls the kernel function *terminate*, which exits the operating system and returns to DOS. If a command the user types doesn't match either of these, the shell will create a subprocess to execute that command as a program. A sample dialogue with this shell is given in Listing 1.

It's easy to add a command to this shell. All you have to do is add another *else if* clause to compare the input with the new command name, and appropriate code to process the command. For example, a *newuser* command could be added to the example shell with only a few lines of code. All it has to do is create a process without the PSL_SUBPROC bit set, and give a communications port as the standard input device. This allows another user to access the system from a remote terminal.

```
C>example

Welcome to the example shell.

>PS

Our process ID is 3
The priority is 5

>myprog
This is a sample program running
under the example shell.  This
program happens to be written in
C, but it could have been written
in any language.

>EXIT
C>
```

Listing 1. A sample of dialogue with the example shell.

When you write your own shell, you can decide exactly how you want the commands to work. If you want, your *dir* command can just accept filenames or directory names like DOS does. You can just as easily write the *dir* command to accept complicated options for selecting files, like VAX/VMS does. Or you can call it *ls* and have it accept different one-character switches, like UNIX.

Next month, we'll get into the internals of Operating System Toolbox. If you'd like to learn more about operating system design, or if you'd like to build your own custom operating system, get a copy of Operating System Toolbox from Wendin today. For examples of more complex shells, you can buy PCVMS or PCNX, both of which demonstrate the power and flexibility of Operating System Toolbox.

Operating System Toolbox: $99
Wendin, Inc.
P.O. Box 3888
Spokane, WA 99220
(509) 624-8088

(continued from page 10)

grams that indirect hashing consistently outperforms such often discussed techniques as binary trees as a means for organizing data that is accessed by name (that is, by the value of a string associated with the data).

Although not explicitly stated, the techniques described by Floyd and Holub are types of indirect hashing; indirect hashing consists of the use of a fixed-size array of pointers (the hash table) pointing to chained lists of buckets, each bucket containing a data item and its name. The buckets are allocated dynamically from a heap. The name of each bucket in a given chain hashes to the index of the array element that points to the beginning of the chain. (Floyd incorrectly defines buckets as the elements of the hash table array.)

Any bucket is reached by getting a pointer to the first bucket in a chain from the hash table and then traversing the chain from the first bucket. It is thus easy to keep a pointer to the previous bucket as each bucket is probed. Therefore, it is not necessary to associate a back pointer with each bucket, as Holub does.

Because only one pointer need be associated with each bucket, the space overhead associated with indirect hashing can be held to H + N pointers, where H is the hash table size and N is the number of data items (that is, the number of buckets). The average number of probes is (1 + N)/2H. The space overhead for a binary tree with N data items is 2N pointers, and the average number of probes is at least log(N/2) (base 2 logarithm).

As a specific example, with 1,000 data items and a hash table size of 250, the indirect hashing technique's space overhead is 1,250 pointers and the average number of probes is 3. The binary tree's space overhead is 2,000 pointers and the best average number of probes is 9.

Floyd states that hashing is effective when ordering (for example, alphabetical ordering) is not important. In fact, I have been using a hashing algorithm that permits alphabetical ordering: the hashing algorithm consists of taking 256 times the ASCII value of the first character of the name, adding the ASCII value of the second character, and dividing the result by 16. (This works with a 4,096-element hash table.) Each chain of buckets is maintained in alphabetical order by inserting each new bucket at the proper place in the chain.

To read out all data items in alphabetical order, then, you simply loop over the hash table elements and, for each nonnull element, read out the buckets of the chain it points to as the chain is traversed. Our products—Source Print and Tree Diagrammer—both use this hashing technique. The simple hash code gives rise to a moderate amount of clustering, but I do not think the clustering is severe enough to seriously degrade probing speed. In fact, Source Print can index all variables within a source code file at about 6,000 lines per minute, and very little of this time is spent dealing with the hashing operations.

Larry R. Miller
Aldebaran Labs.
3339 Vincent Rd.
Pleasant Hill, CA 94523

Baby Ducks

Dear *DDJ*,

The wish list for the ultimate editor ["Text Editors," February 1987] surprised me a bit, as the one I use for word processing (simple stuff, such as this letter) and mostly for programming has virtually everything on the list. The package is KEDIT from Mansfield Software Group. It has everything from the Must Haves list, and then some. In the Not Necessary . . ., it also covers all. In fact, in the entire list of wishes, the only thing it cannot do is edit files larger than memory.

I program in C, and I have macros to do things such as comment or uncomment a line (same key) and my Can't Live Without: given an *if* or *for* (*while, until* . . .) statement, I press a key combo (Ctrl-F2) and it indents and puts an open bracket, blank line, and close bracket followed by a comment (the first 20 characters of the *if* or loop statement), then positions me at the first indented position of the blank line. I go from:

```
for(i=0; i<100; i++)
```

to:

```
for(i=0; i<100; i++)
{
} /* for (i=0; i >100; i++). . .*/
```

in one keystroke.

KEDIT has many other features, and for $125 it is one of the best buys around. No, I don't sell KEDIT or know the firm, but I love to share knowledge about excellent software with others.

Flip Nehrt
1209 N. Topeka
Whichita, KS 67214

Dear *DDJ*,

I read with great interest the article on word processing [February 1987]. I agree with Thomas and Turner about the baby duck syndrome. I wish to offer a wish list in reply:

1. Text buffered to mass storage. Wait time should be minimal. Anticipatory loading and dumping could be implemented.
2. Response to keyboarding should be adequate, so fast typists do not feel impeded by the computer.
3. Reduced command set that works quickly.
4. Wordwrap and left and right justification.
5. Screen should display the printed page. Special fonts such as italics are unnecessary. After all, who wants to change a daisy for two words. Cute code is expensive.
6. Block move, copy, delete, fetch, and store to disk.
7. Search and replace globally with case insensitivity.
8. Multifile and multiwindow.

What I really want is an editor that can be tailored to the task at hand.

As the good Doctor has always been at the forefront of new developments—providing us with Tiny BASIC and Small-C—perhaps it or its readers could provide us with an editor that is machine independent and extensible.

Robert B. McCormick
11 East Chestnut St.
Bordentown, NJ 08505

DDJ

Why Are So Many People Switching to Smalltalk/V?

Why are scientists, engineers, and professionals switching to Smalltalk/V? Because it lets them do amazing things on their PCs, with a Mac-like interface and an easy-to-use object-oriented language. And with Smalltalk/V you get an unsurpassed array of problem-solving tools. You can even personalize the entire system to suit your needs.

Smalltalk/V is the programmable environment that gives you total control of your computer and makes it what it was meant to be, a truly personal tool for your mind.

"Smalltalk/V is the highest performance object-oriented programming system available for PCs."

Dr. Piero Scaruffi, Chief Scientist, Olivetti Artificial Intelligence Center

"Smalltalk/V is an excellent buy and makes a good alternative to other programming languages for the development of complex applications."

Bill Wong, Director, PC Labs PC Magazine

$99.00

Smalltalk/V

The Programmable Environment

"This is the real thing, folks. A super Smalltalk like this turns your PC into a hot workstation. It's fantastic . . . Highly recommended."

John C. Dvorak, Contributing Editor, PC Magazine

"My background is in physical chemistry, not in programming. I like Smalltalk/V because I can use objects in the computer to represent objects in the physical world."

Dr. Paul Soper, Senior Specialist E. I. du Pont de Nemours & Co.

"Smalltalk/V is a productive programming environment that allows us to quickly develop sophisticated medical applications."

Dr. Mike McCoy, Dean for Instructional Computing UCLA School of Medicine

"Smalltalk/V, with its visual interface and class structure, is a perfect way to simulate the complex interactions of natural systems."

Lee A. Graham, Research Assistant Institute of Ecology, University of Georgia

"I solve problems quickly using Smalltalk/V because its classes and objects help me organize my thinking. And besides, it's fun to use."

Dr. Barry Fishman, Sr. Project Engineer Hughes Aircraft Company

BYTE and BIX are trademarks of McGraw-Hill, Inc. IBM, IBM-PC, and IBM PC-AT are trademarks of International Business Machines Corporation. Unix is a trademark of Bell Laboratories.

Other Smalltalk/V Features

- Object-oriented Prolog integrated with the Smalltalk environment
- Supports exploratory programming and prototyping
- Class hierarchy with inheritance creates highly re-useable source code
- Smalltalk source code included, with browser windows for easy access and modification
- Object-swapping creates a virtual memory on hard or RAM disk
- Bit-mapped graphics with bit and form editors
- A sophisticated source-level debugger
- Automatic change log for easy recovery from errors
- Powerful directory/file browser system for organizing DOS files
- Access to other languages and DOS functions
- 500 page manual with comprehensive tutorial and reference sections
- Optional add-on modules
  - RS-232 communications interface with UNIX™ and TTY windows
  - EGA color graphics
  - "Goodies" diskette, including multiprocessing, music, zoom, object loader, and more

Yes! I want to turn my PC into a hot workstation! Send me . . . .

☐ Smalltalk/V-The Programmable Environment $99
☐ Communications Option $49
☐ EGA Color Option $49
☐ "Goodies" diskette $49

Shipping and Handling .... $________

CA residents add applicable sales tax .... $________

TOTAL .... $________

Shipping and Handling
U.S., Canada, Mexico .... $ 5.00
Elsewhere $15.00

I enclose ☐ Check ☐ Money Order
☐ Credit card information ☐ MC ☐ VISA

Number: ________________
Expiration: ________________
Signature: ________________
Name: ________________
Street Address: ________________
City/State/Zip: ________________
Phone: ________________

NOT COPY PROTECTED, 60-DAY MONEY-BACK GUARANTEE
ON-LINE USER-SUPPORT CONFERENCE ON BYTE'S BIX™

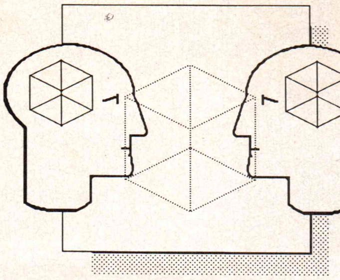
Smalltalk/V requires DOS and 512K RAM on IBM PCs (including AT) or "compatibles," and a CGA, EGA, Toshiba T3100, Hercules, or AT&T 6300 graphic controller. A Microsoft or compatible mouse is recommended.

digitalk inc.

5200 West Century Boulevard
Los Angeles, CA 90045 (213) 645-1082

CIRCLE 127 ON READER SERVICE CARD

VIEWPOINT



Things Computers Can Never Do

Anyone who has witnessed the enormous improvements in computers in the last 40 years may get the impression that computers will eventually be able to solve every well-defined problem. Progress in language understanding and other forms of artificial intelligence has been disappointing, but human language is full of ambiguities, so that's not a well-defined problem. Chess, on the other hand, is very well defined. Although it was once considered the epitome of intelligent activity, computers can now play chess better than all but a few human players.

Some problems, although well defined, are too large to be solved in a reasonable time even on our largest computers. But surely, if a computer could be freed from all limitations on time and memory, couldn't it solve any well-defined problem?

The surprising answer to this question, which was known to mathematicians even before the first real computers were constructed, is no. There are some things no computer can ever do because it can be proved that

by Philip J. Erdelsky

there are no algorithms to do them—just as there is no way to square a circle with a compass and straightedge.

These things are not mere mathematical curiosities. They are things that programmers would like to have their computers do for them and things that the suppliers of software development tools would like to incorporate into their debuggers. Computer science curricula usually include the subject of uncomputable functions, but programmers who are not computer science majors sometimes ask for the impossible without realizing it.

Alan Turing in 1935 asked whether there is a method by which a computer program can determine whether any other computer program will halt. This is the famous "halting problem." Turing showed that it has no solution.

A debugger with this ability would certainly be useful. Failure to halt normally is a common form of program failure. Moreover, the debugger could be applied successively to parts of the failed program to isolate the part that is hanging up.

It is not obvious that such a debugger is impossible. Of course, the debugger can't just single-step the program to see if it halts. If the program doesn't halt, the debugger could run forever without determining that this is the case. Or it might give up just as the program is about to terminate, as human programmers sometimes do. At some point, the debugger would have to be able to say, "Aha! This loop is infinite!" It seems as though a cleverly written debugger, having all the tools of modern high-level languages at its disposal, might be able to do that.

The impossibility proof is based on the following argument. If you have a debugger that can solve the halting problem, given unlimited time and memory, then you can use the same code to make the debugger do other things, some of which are self-contradictory and hence impossible.

The particular computer language is not important. If you can solve the halting problem for one language, you can solve it for another. Just use a compiler or other translation program before solving the halting problem. Notice that translating an assembly-language program to a higher-level language is quite easy, although the object program is bound to be inefficient. The goal, however, is to show that a solution to the halting problem is impossible, not merely inefficient.

Turing himself proposed a minimal machine that has come to be called the Turing Machine. Its memory was supposed to be infinitely long but only one bit wide, and the machine had only sequential access to it, as with a tape. The programming language was essentially a flowchart, with only a few basic commands. Nevertheless, Turing showed that his machine was able to emulate any other machine, given enough time and a suitable program. Such a construction is not necessary for our purposes—you can imagine that the computer is programmed in some familiar high-level language.

Now consider the problem of determining whether a program can print out a specified string *S* (with or without other output). If you can solve the halting problem, you can solve this problem. Just replace every print statement in the program with a routine that does not send the output to the printer but keeps track of the output and halts when the string *S* appears. Then, to keep the program from halting for any other reason, replace all the halt statements in the program with endless loops. Then solve the halting problem for the result.

Such a program would be useful in itself because many run-time errors produce distinctive messages, and it would be helpful to predict in advance that such errors will occur.

Because this applies to any string S, you can also determine whether a program prints out a copy of itself. This is not as curious as it appears at first glance. It is easy to write a 1,000-character program that prints out all combinations of 1,000 characters including itself. In fact, 1,000 characters is probably an over estimate of the number of characters required in most high level languages.

Now you can write a program to do the following things. First, generate, one by one, all possible programs. The easiest way to do this is to generate all strings and check each one to see whether it is a program. Compilers do this when they check syntax. Then check each program to see whether it prints out a copy of itself.

(continued on page 140)

Philip J. Erdelsky, Data/Ware Development Inc., 4204 Sorrento Valley Blvd., San Diego, CA 92121. Philip is a software manager.

BEWARE OF THE AT MEMORY GREMLINS.

ONLY ECCELL™—THE ECC-PROTECTED MEMORY CARD—CAN KEEP YOU FROM BECOMING THEIR NEXT VICTIM.

If you're a serious AT user, you live in constant fear of seeing this memory error message:

That's your friendly AT's way of saying, "All your long hours of hard work just went down the tubes."

About then you start suspecting gremlins in your AT's memory. The fact is, these errors can be caused by a *single* bad RAM bit out of a possible *256 million*.* The odds are against you.

Whatever the cause, these memory crashes become more common when you add a lot of AT memory or leave your system on day and night—like you do for network file servers, bulletin boards or host emulation.

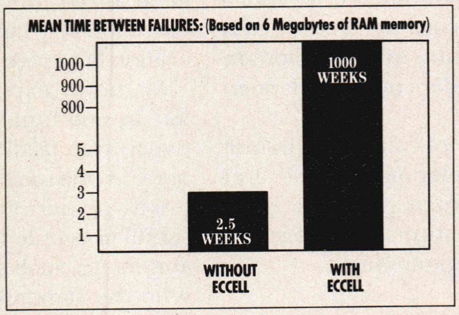
*Based on 16 megabytes of RAM.

Now the good news. One memory card offers a solution to this potentially disastrous problem: Orchid's ECCELL.™ And *only* ECCELL.

KEEP WORKING WHEN THE CHIPS ARE DOWN.

With ECCELL, you're protected against losing all your valuable work due to AT memory crashes. Only ECCELL uses an ECC (Error Correction Code) mechanism to continuously check for errors—and actually *correct* them before they can do any harm. *No other AT memory card does this.*

ECC protection has been available on mainframes and minicomputers for years. Now ECCELL brings this sophisticated technology to the AT user.



HANDLE DOS, PROTECTED MEMORY AND EMS.

ECCELL gives you conventional DOS memory up to 640K. Extended memory that supports protected mode DOS. And expanded memory conforming to the Lotus®/Intel/Microsoft® EMS standard. Plus ECCELL can be used with other AT memory expansion cards.

DON'T PAY EXTRA FOR I/O PORTS YOU DON'T NEED.

If you need to connect more peripherals, ECCELL offers optional serial/parallel or dual serial ports. Unlike other cards, if you don't need them, you don't pay for them.

ADD 3 MEGABYTES OF RAM PER CARD.

Using multiple ECCELL cards, you can install up to 12 Megabytes of ECC-protected RAM. Installation takes only minutes, guided by an intelligent set-up program.

So don't become the next victim of the AT memory gremlins. Protect yourself with ECCELL. Call **(415) 490-8586** today. Or contact your local dealer. ECCELL is a trademark of Orchid Technology. All other product names are trademarks of their manufacturers.

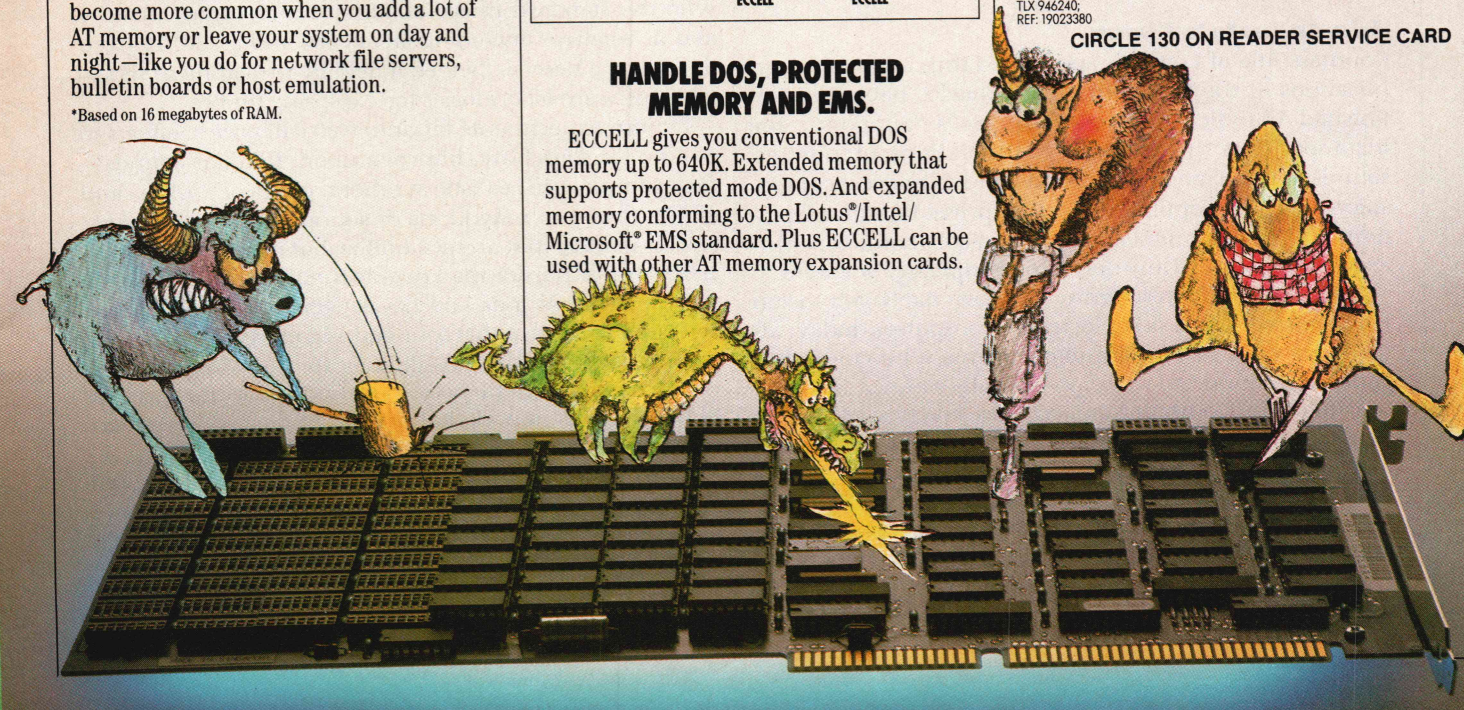
ECCELL™

ORCHID TECHNOLOGY
45365 NORTHPORT LOOP WEST
FREMONT, CA 94538
(415) 490-8586; TLX 709289.

ORCHID (EUROPE) LTD.,
UNIT 9A, INTEC-2,
WADE ROAD,
BASINGSTOKE, HANTS,
RG24 ONE,
GREAT BRITAIN;
TEL 0256-479898;
TLX 946240;
REF: 19023380

ORCHID

CIRCLE 130 ON READER SERVICE CARD



Pushing the Sound Envelope

by David Levitt

This year heralds an exciting time for music software. Developments during the past few years have pushed the first grand visions of music software enthusiasts rapidly toward reality. Moreover, the wide adoption of MIDI and decreasing hardware costs mean that discoveries move quickly from laboratories into recording studios and homes. In fact, today's music algorithms originate both in research labs and in the dens of readers of journals such as this one.

These programs make it easier for musical novices to make music.

This article includes a brief history of computers in music and then focuses on recent developments in several areas: MIDI, sampling, transient-oriented synthesis methods such as the Karplus-Strong algorithm, and programs that compose and collaborate on original music.

Ada Augusta's Vision

Countess Ada of Lovelace speculated thus on musical applications of the first computer—Charles Babbage's unfinished Analytical Engine—in the 1840s: "Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the Engine might compose and collaborate scientific pieces of music of any degree of complexity or extent."[1]

It took more than a century before Babbage's vision was realized in the first electronic computers; today, after more than 30 years of experiments with computer-generated sound, Ada's vision is finally close at hand. In the interim, "computer music" has taken on the role played by atonal and "experimental" music in the first half of this century.

Timbre

In the 1950s, Max Mathews of Bell Laboratories was among the first to explore the computer as a generator of as yet unheard sounds. At night, Bell Labs' computers became generalized timbre generators, and Mathews' FORTRAN-based MUSIC V language became the first digital signal generation language intended for composers.

Mathews experimented with composing algorithms, too. In one funny piece the computer "interpolated" between two traditional melodies, gradually replacing the notes of one melody with notes from the other. Still, MUSIC V was primarily a timbre-generation language; it wasn't useful to people who wanted to deal primarily with pitch and meter, such as professional composers and musicians who use standard music notation. For decades, MUSIC V and its relatives (including MUSIC 360 and MUSIC 11 from MIT's Barry Vercoe) defined a field by providing semiportable software laboratories for exploring timbre.

Timbre experiments fall into two categories: efforts to imitate (or somehow, improve upon) familiar sounds—for example, the realistic synthetic piano or violin—and efforts to create new kinds of sounds, or transitions between sounds, that are unfamiliar but intrinsically interesting. It was harder than people thought to approximate some natural sounds and not especially easy to build a synthetic orchestra that sounded as good as a real one. On the other hand, the possibilities for new sounds, and pieces based on them, beckoned. Computer music quickly became the new digital branch of electronic, experimental music.

This trend continued into the 60s and 70s when Stanford's Center for Computer Research in Music and Acoustics (CCRMA) laboratory gained prominence. John Chowning, cofounder of CCRMA (pronounced "karma"), showed

David Levitt, 117 Harvard #3, Cambridge, MA 02139. David is a research scientist in the Entertainment Group at the Massachusetts Institute of Technology's Media Laboratory.

that frequency-modulating a simple audio signal with a second signal in the audio band resulted in rich timbres—the idea that became Yamaha's DX/TX FM synthesizers almost 20 years later. More recently, CCRMA scientist Marc LeBrun generalized the FM algorithm, creating a more powerful method known as nonlinear waveshaping. In the 70s, former CCRMA students such as Andy Moorer and F. Richard Moore founded the IRCAM music research lab in Paris, Lucasfilm's audio lab, and University of California at San Diego's computer music program. These groups developed special signal-processing hardware to synthesize timbres quickly, often in real time.

Despite such advances, many people still liked sounds made by banging, plucking, and blowing into physical objects as much as or more than synthetic sounds. This wasn't simply a matter of familiarity or cost; something was missing from most of the synthetic sounds and still is.

Gesture and Graphics

Part of the problem is the absence of gesture. Today's timbre software is best at controlling the loudness, pitch, and timing of a sound *independently* of timbral parameters. In natural instruments, the timbral details are often *nonlinearly coupled* with loudness, pitch, and even timing. How would you describe the different ways and speeds in which a violinist can move a bow? Or the different ways a flautist blows a flute? Most software doesn't try. Not only do most input devices have too few degrees of freedom but also the software doesn't provide easy access to the right parameters.

Other natural instruments have more high frequencies when you strike them harder. So even a percussive instrument such as the piano has a nonlinear, velocity-coupled component that will be missed in a synthetic instrument with a linear loudness model. The gestural subtleties in natural instruments are poorly understood, but evidently they are important to many listeners, and they make it hard for synthetic timbres to compete.

Gesture has always been a central concern for William Buxton and his students at The University of Toronto. In the 70s, Buxton's lab gained prominence by focusing on the quality of interaction with the computer while making music with it. Buxton's work made extensive use of pointing devices, graphics, and other innovations.

In the same period, Alan Kay's group at Xerox PARC, which also held user interaction sacred, made a music system with FM, graphical timbre design, and simple score editing. This was followed by Mockingbird, an interactive music editing program by John Maxwell III and Severo Ornstein, also from Xerox PARC. Mockingbird allowed users to edit a rough transcription of a keyboard performance with the mouse, using traditional music notation. Its capabilities still exceed those of today's commercial music editing software. Meanwhile, Don Byrd at the University of Indiana produced SMUT, the most versatile program for printing standard musical notation.

Transients

Lack of gesture is only part of the problem with today's synthetic timbres. Most timbre algorithms provide control over the steady-state frequency spectrum of the sound, but that is not the best way to control its subjective effects. Listeners are especially sensitive to transients—short (50 milliseconds or less) periods of sudden change—especially in the initial attack. We have no trouble recognizing a piano on cheap speakers, though most of the low and high frequencies are missing, because the changes in the parts we do hear are correct. In fact, experiments show that, if a steady violin tone is preceded by the attack from a trumpet, most people hear a trumpet. Transients dominate. But traditional timbre creation methods such as additive synthesis (adding up harmonics with different amplitudes) and FM don't provide direct control over them.

In 1983, Stanford student Alex Strong invented a simple method for controlling attack transients. He didn't use the fancy signal processors from up the hill at the CCRMA lab; instead he hunted for a way to make more interesting sounds on his 8080 homebrew computer, the size of a

MUSIC SOFTWARE
(continued from page 17)

shoebox. He toggled in an 8080 program (the machine had no keyboard), and in real time it generated what sounded just like a plucked guitar string.

Strong teamed up with Kevin Karplus, another Stanford student, to create the Karplus-Strong algorithm and a new approach to synthetic timbre design. The algorithm is simple: one cycle of the sound is stored in a memory buffer and is played repeatedly through a digital/analog converter. The pitch is determined by the size of the buffer and the rate at which samples are played:

pitch=samprate/bufsize

This is the way wave tables work in many simple synthesis programs. But with Karplus and Strong's scheme, each time the waveform is played, a filtered version is computed and stored back where the previous waveform was. Thus the algorithm defines a feedback loop in which a filter is used over and over to transform the waveform data.

If the buffer waveform is initially white noise, and the filter is a simple, first-order low-pass filter, we have Strong's basic guitar algorithm. Intuitively, we see that each time through the loop, more and more of the high frequencies are filtered out, so the sound changes quickly from a sharp scratch to a pure tone at the frequency samprate/bufsize. This is the plucklike guitar attack. Strong implemented the low-pass filter by averaging adjacent samples—simply adding and right-shifting pairs of samples on the 8080—and was able to produce two guitar voices in real time with no special hardware. With slight changes in parameters, the algorithm also produced convincing banjo sounds.

Algorithms such as Karplus-Strong have not yet been fully explored. Still, the essential idea is important: designers can gain control over the attack and other transients by creating a filter that describes relative *rates of change* in the sound's frequency components. A low-pass filter in the feedback loop means the higher frequencies fade, as in a "pluck" attack; a high-pass filter means the higher frequencies grow, as in a horn or other brass instrument attack. (I am not aware of attempts to produce brass sounds with Karplus-Strong; this is a research topic.)

So, timbre designers are still learning new ways to create synthetic sounds that listeners find realistic or appealing. Some day we may yet have synthetic sounds that unquestionably improve on their physical predecessors.

Sampling

Until then, we're being rescued by falling memory costs. If we can't easily improve on natural sounds, we can record them into digital samples, then resample (scale the frequency) and loop, playing them back with a different pitch, loudness, and duration. The digital recording means the whole sound—including transients—is faithfully captured. We still have to account for nonlinear scaling, usually by recording several samples over a range of pitch and loudness values. To make an instrument based on a human voice, for example, we sample several voices in different ranges; then when we play a high note, it will sound like a soprano, not like Alvin the Chipmunk.

Sampling memory is at the heart of the Kurzweil, Emulator II, Prophet 2000, and many other recent synthesizers; the samplers are sure to decrease in cost and increase in popularity. Today 8-bit samplers are available for less than $2,000. Within a few years the 16-bit, 44.1-kHz per channel format used in audio compact discs is likely to become the standard for computer sound sampling, too.

MIDI

Of all these innovations, the introduction of the MIDI (Musical Instrument Digital Interface) standard in 1984 is having the greatest effect on modern music software. MIDI has several limitations, covered in more detail in Mark Gavin's MIDI article in this issue, but the very existence of an interface standard has helped programmers focus on issues that are independent of the synthesis technology—that is, on music composition and user interaction rather than timbre design.

On the plus side, MIDI has the necessary rudiments for gestural control: velocity and aftertouch for keyboard instruments and other linear control options (for example, for pedal and breath control). MIDI even includes space for instrument makers to invent their own new codes. MIDI's demerits include no standard codes for polyphonic pitch bending (so most MIDI synthesizers have a fixed chromatic tuning) and a 31.25-kilobit/sec data rate that is simply too slow for some dense polyphonic pieces.

Despite its flaws, MIDI is making event-level recording and computer control the rule rather than the exception in new instruments. Companies such as Yamaha have even begun to include MIDI outputs in their acoustic pianos.

This means a welcome end to computer music, which we needn't distinguish from "real" music for much longer. Presumably other media suffered an early period in which the technology was so peculiar it dominated the perceptions of both artists and audiences. Still, we don't consider the hardcover novel a separate art form, and it's hard to imagine a book so dull that our first comments would be about the paper it's printed on. Now that virtu-

ally every music studio has a MIDI synthesizer and will soon have a computer, computer music will fade into the past and innovative composers will find new ways to distinguish themselves. Thus MIDI is nudging us nerds away from timbre design and toward composition software —the rhythm, melody, and harmony problems that have remained the backbone of music theory, the music business, and musical communication.

Algorithmic Composition

The rest of this article touches on the current state of this exciting musical field: representing the knowledge of the composer and improvisor in software. This is what the countess dreamed of when she wrote that "the Engine might compose and collaborate scientific pieces of music. . . ." Algorithmic composers write programs that determine what notes to play and when. This has little in common with timbre design; algorithmic composers might even dislike synthetic timbres, preferring to print the scores their programs generate so they can be read and performed on natural instruments by human musicians. Thus, algorithmic composers are a small, almost separate subculture of music programmers—often working independently in the same laboratories as their timbral counterparts.

I have been writing programs that improvise jazz and arrange a melody in a given musical style for about ten years. My graduate work in the MIT Artificial Intelligence Lab entailed writing LISP programs that represent chords, melodies, and relationships between them—much as I do when I improvise jazz, invent a new exercise, or solve a musical problem at the piano. I have developed style templates that use a given melody and chord progression and create simplified parodies of bass players, bebop, ragtime, and New Orleans jazz ensemble improvisation.

Others in the field include Fry, who also wrote jazz improvisation programs at MIT. His most advanced programs could write an additional chorus for John Coltrane's famous Giant Steps solo or Iron Butterfly's "In a Gadda da Vida." David Wessel, now in Paris at IRCAM, writes programs that produce convincing, often overimaginative, blues solos. Peter Langston of BellCOR has produced three-part rock harmonies and pieces based on organic "growth" algorithms.

Laurie Spiegel was a composer/scientist in Max Mathews' area at Bell Labs and today makes her living as a composer in New York City. Her programs reflect her interest in folk melodies and traditional harmony. Spiegel's programs are sufficiently subtle and musical that people assume her pieces weren't written with a computer.

Bill Schottstadt developed the Pla composing language at CCRMA, where he and others use it for algorithmic composition of relatively traditional pieces. For instance, when CCRMA student David Jaffc extended the Karplus-Strong algorithm to simulate a particularly realistic banjo and guitar, he used Pla algorithms to build a parody of Kentucky bluegrass arpeggios in an arrangement he called "Silicon Valley Breakdown."

Roger Dannenberg of Carnegie Mellon University has been pioneering a different area: software that accompanies a live performer, following the tempo and playing all the other parts. Barry Vercoe and Miller Puckette of MIT have a similar program, which also learns to anticipate the performer on subsequent rehearsals.

In short, we are seeing a renaissance of knowledge-based music composition and collaboration software. Although most of these projects are based in university research labs, the software can also run on personal computers. For instance, we have several such programs running on the Macintosh at the MIT Media Lab. Moreover, several algorithmic music software titles have appeared on the market in 1986 and 1987.

In Laurie Spiegel's Music Mouse for the Macintosh, a performer controls the overall motion of up to four voices using the mouse, while the program fits the performer's gestures into a selected scale (for example, Major, Pentatonic, or Middle Eastern). Notes are also constrained to change only on metrical time boundaries, so every gesture is automatically both tonal and rhythmically "musical." Typewriter keyboard switches provide control over a range of other musical parameters.

In Bob Campbell's Instant Music for the Amiga, the performer can control pitch with the mouse while seeing harmonic and rhythmic information on the screen. Important harmonic relationships are indicated cleverly by color. In one mode, you collaborate with the computer on a lead line: you control pitch while the program controls rhythm. When your ideas conflict, you can hear and almost feel the tension—again without losing "musicality."

David Zicarelli's Jam Factory, also for the Mac, segments your MIDI performance into phrases and plays them back—either with new accents or stochastically scrambled (using a Markov algorithm) into a sort of musical wallpaper that's like what you played. For each of up to four performances, you control about a dozen playback parameters with the mouse and can play along with them.

Zicarelli also contributed to Joel Chadabe's M, an interactive composing and performing system. Chadabe has been making composing algorithms for years; I haven't had a chance to try his program yet.

Most of these programs make it much easier than ever for musical novices to make a piece of music that sounds good. The knowledge in the program provides a substitute for musical virtuosity. As computer scientist Alan Kay likes to ask: Is it an intelligence amplifier or just a prosthetic? To the author of the program, it's an amplifier; she or he already knows the musical concepts and is controlling them with greater leverage. For some users, it might be a crutch; but programs that provide graphic or other feedback can show novices what the computer is doing and what knowledge it is using. Then the programs can be more like "training wheels," temporary stabilizers that can eventually come off. Ultimately, users themselves will decide.

Notes

1. Robert Taylor, ed., *Scientific Memoirs, Vol. III: Ada Augusta, Countess of Lovelace* (Johnson, 1943).

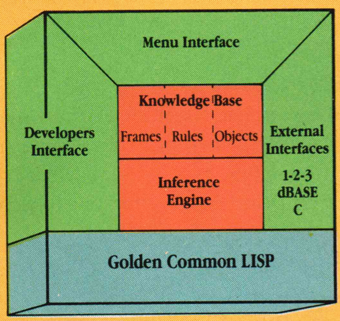
DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 2.**

Now you can build and deliver serious expert systems on advanced PCs.

Introducing GoldWorks, formerly code-named Acorn.

GoldWorks is Gold Hill's premiere product for serious expert system building on 286- and 386-based PCs. It's designed for professional software developers who need to integrate expert systems with conventional applications running on conventional hardware. And it combines the best features of high-end expert system tools into a new standard for expert system development and delivery on advanced PCs.



GoldWorks is the most comprehensive expert system tool available for advanced PCs.

Works like a shell.

GoldWorks gives you the best features of an expert system shell. With the easy-to-use menu interface, you can rapidly prototype and build expert system applications without knowing the underlying programming environment. And you get the GoldWorks tutorial, the San Marco LISP Explorer® tutorial, an on-line help system, and example applications to get you started quickly.

Works like a toolkit.

GoldWorks gives you the best features of an expert system toolkit. You can access the underlying programming environment to extend and customize the system for your specific applications. And you can address up to 15 MB of extended memory on the PC AT (and even more on 386-based PCs).

Works like expert system tools previously available only on high-end workstations . . . at a fraction of the cost.

GoldWorks sets a new standard for expert system development and delivery on advanced PCs. You get frames with multiple inheritance for flexible knowledge representation. Rules supporting integrated forward and backward chaining for powerful inferencing. Object programming for developing modular applications. Plus advanced features for controlling the inferencing process, including rule sets, sponsors, rule priorities, certainty factors, and extensive rule inspecting and debugging facilities. All on conventional hardware—the PC you already use.

Works to develop <u>and</u> deliver your expert systems.

GoldWorks is the <u>only</u> tool that lets you develop and deliver serious expert systems on PC ATs. And **GoldWorks** also takes advantage of PCs based on Intel's powerful 80386 processor, including the COMPAQ DESKPRO 386 and Gold Hill's 386 LISP System.

Works to integrate expert systems with conventional PC applications.

With **GoldWorks**, you can integrate expert system applications with dBASE III and Lotus 1-2-3 . . . integrate C routines and libraries into your expert systems . . . and build and deliver expert systems in network environments.

Works the way you want an expert system builder to work.

GoldWorks from Gold Hill sets the standard by which all other expert system tools will be measured. It was extensively field-tested by developers in dozens of major corporations. And GoldWorks is backed by Gold Hill's comprehensive customer support, training and consulting programs.

Now you have the expert system builder that works the way *you* want to work—**GoldWorks.** To see how it works, order our unique Demonstration Kit, including full color video and complete User's Guide. It's only $49 postpaid, refundable with your **GoldWorks** purchase. To order, call toll-free:

1-800-242-5477.
In Mass., call (617) 492-2071.

GoldWorks from Gold Hill.
The expert in AI on PCs.

Gold Hill Computers, Inc.
163 Harvard Street
Cambridge, MA 02139

GOLD HILL™

CIRCLE 291 ON READER SERVICE CARD

Designing a Music Recorder

by Mark Garvin

Here are ways to get started writing your own MIDI software.

Imagine designing products for a changing microcomputer market without hardware and software standards. Standard architectures such as IBM PCs and Apples have led to a proliferation of new microcomputer products, and the development of MIDI (Musical Instrument Digital Interface) has resulted in a similar revolution in the music industry. By providing common ground for communication, MIDI allows software engineers and musicians to access a wide range of synthesizers, computers, and musical controllers.

MIDI has become virtually uncontested as a means to link synthesizers and computer equipment and is now finding acceptance in many related industries such as lighting control, film editing, and automated audio mixing. By supporting real-time access to so many devices, MIDI has opened new dimensions for recording, composition, and live performance. Now musicians can generate orchestral scores with banks of rack-mounted synthesizers, coordinate sound and visual effects, and even transmit stored musical data over phone lines.

In this article I outline several applications of MIDI and I suggest several ways to get started writing your own MIDI software. I have listed some of the current products and manufacturers, but these listings are by no means complete. They are given only to provide a starting point for obtaining additional information.

Mark Garvin, Xymetric Productions, 211 W. Broadway, New York, NY 10013. Mark has worked in both music and electronics for 15 years. He recently designed an eight-port MIDI controller that runs on the IBM PC/AT.

Overview

The MIDI specification has remained reasonably intact since it was first proposed by synthesizer manufacturer Sequential Circuits in 1982. It is sufficiently specialized to handle most direct musical communication, yet its generality has allowed it to adapt to applications that were not foreseen when it was originally drafted.

MIDI entails both a hardware and a software specification: the hardware consists of a relatively fast optically isolated serial loop with separate cables for send and receive; the software provides detailed methods for transmitting note-control data and looser specifications for handling interaction between products from different manufacturers. MIDI works in much the same way as an RS-232 modem protocol, but it is optimized for musical data. Modern MIDI record/playback systems could be regarded as a type of multiprocessor network because an intelligent master (keyboard or computer) controls a series of devices, each with its own onboard intelligence.

MIDI commands include *NOTE-ON* and *NOTE-OFF*, response *MODE* for filtering received signals, *REAL-TIME* messages for coordinating events, and *SYSTEM COMMON* and *EXCLUSIVE* commands for setting up songs or addressing a particular brand of synthesizer. This simple instruction set allows enough flexibility to accommodate most synthesizer architectures while providing much needed universal music commands. As more manufacturers have realized the advantages of communicating with a wide array of musical equipment, MIDI's popularity has mushroomed. Few synthesis instruments are sold today without MIDI interfaces.

All MIDI documentation is now handled by the International MIDI Association (IMA). (See the box on page 48.)

Products

Fortunately, the high level of competition among musical instrument manufacturers has been offset by specialization: small companies can offer transmit-only devices, such as high-quality keyboards with no sound output, or receive-only devices, such as sound generators that respond only to MIDI input. Undoubtedly, the broadest new field is that of software-based controllers. These usually connect between a keyboard (or other source) and a sound generator, where they monitor and control MIDI communication. Some of the newer MIDI-based products include guitar, voice, and even xylophone-to-MIDI converters (Roland, Fairlight), software-hardware retrofits for playing digitized notes from personal computers (Hybrid Arts), MIDI-controlled reverb and echo units (Lexicon, KORG), and MIDI-controlled audio mixing consoles (AKAI).

Synthesizers

Early synthesizers—used for scoring

so many old science-fiction movies—were assembled from several modules that were interconnected manually by patchcords. These and other voltage-controlled instruments have now attained a certain vintage status.

New instruments use computerized signal routing, and modern sound-generating techniques range from additive synthesis (Kawai), FM synthesis (Yamaha), and phase-distortion synthesis (Casio) to actual digital recording, or sampling, of natural sounds (Sequential Circuits, E-Mu, Kurzweil).

Additive synthesis uses the addition of several sine-wave components to produce an output waveform. Theoretically, any waveform can be broken down into sine-wave components by using a process known as Fourier analysis. It follows, then, that any waveform can be recreated by adding these same sine-wave components. In actual application, the process is not so simple; the human ear quickly becomes bored with the static, or unchanging, waveform that is created. It is this static characteristic of early synthesizers that contributed to the stereotyped monotonous or bland sound.

The waveforms generated by traditional acoustic instruments change as notes are being played, so recreating the natural timbres of these instruments requires real-time control over the amplitudes, or envelopes, of the sine-wave components. In early synthesizers, this was approximated by the use of an envelope generator that cycled through Attack-Decay-Sustain-Release (ADSR) states in response to key-down and key-up events. Patching the ADSR generator into voltage-controlled filters and voltage-controlled amplifiers provided a primitive level of control over the harmonic structure of the waveform. Newer machines sometimes use a separate programmable envelope for each sine-wave component of the waveform. With sufficient control of the amplitudes, any conceivable sound can be recreated without having to use digitized wave samples. This is the objective of the resynthesis or adaptive synthesis machines, such as Roland's digital piano.

FM synthesis uses a limited number of sine-wave oscillators with individual envelopes, so in this respect it bears some resemblance to resynthesis methods. One significant departure is in the way the oscillators are configured: different algorithms can be chosen to allow oscillators to intermodulate, creating rich and sometimes enharmonic frequencies. The resulting output can range from clangorous-sounding bells to human-sounding voices, but FM machines are relatively unpredictable and difficult to program.

Phase-distortion synthesis involves scanning a simple (usually sine) wave

MIDI works in much the same way as an RS-232 modem protocol.

and varying the scan rate as the wave is being replayed. In other words, the leading edge of the sine wave can be scanned rapidly so it appears to be a nearly vertical edge; the trailing edge can be scanned more slowly so it has a tapered slope. The resulting sawtooth waveform is much richer in harmonics than the flute-like sine wave. Dynamic variation of the scan rate can change the shape and timbre of the waveform as a note is being played.

Some of these methods of sound generation may seem to make the use of waveform sampling unnecessary, but in fact samplers can usually do much more than recreate natural sounds. For example, precise (up to 16-bit) digitizations of orchestras, drums, waterfalls, or human voice can be altered and played back at any pitch. The elite of the sampling synthesizers (Fairlight, New England Digital) can cost hundreds of thousands of dollars, but they eliminate the need for a lot of expensive equipment in a recording studio. New England Digital even advertises a tapeless studio that records audio tracks directly to a high-capacity hard disk. Optical Media offers prerecorded sound libraries on CD ROMs. Such systems are becoming more affordable as the cost of mass storage continues to drop.

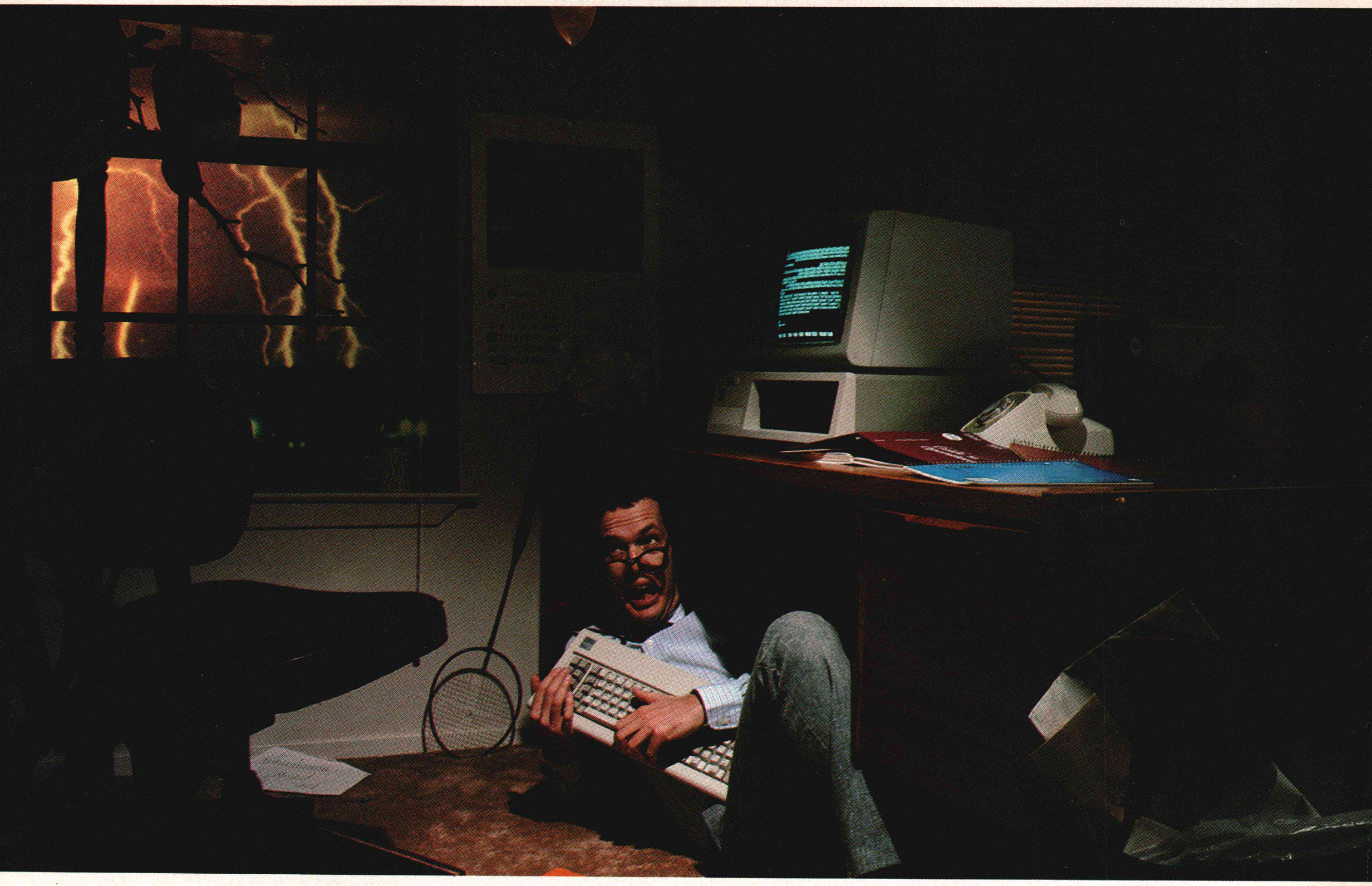
Patch Editors and Librarians

These diverse methods of sound synthesis have introduced a new class of problems: a few potentiometers on a front panel are no longer sufficient to program (that's right, *program*) a synthesizer. One manufacturer claims that, if all of its front-panel functions were to be made available at once, its synthesizer would be 17 feet long. Instead of overlaying the limited set of front-panel controls with multiple modes, synthesizer front-panel functions can usually be accessed by sending a special set of MIDI *SYSTEM-SPECIFIC* commands from a computer outfitted with MIDI ports.

Synthesizer programming has become an art in the same sense as actual performance of music, and album covers frequently give credit to sound programmers, even if they do not perform on the album. The sound programs, which can make or break the sound of a synthesizer, are known as patches. Good programmers frequently make a living solely by selling their patches, either in the form of instruction sheets that simply explain how to recreate the original sounds or in a downloadable binary form on floppy disks. Disk-based systems require the use of a patch librarian program to organize and access patch files. Most patch librarians allow two-way communication between the synthesizer and computer, so patches can be sent to a synthesizer, modified with the use of the synthesizer's controls, and sent back to be archived on disk. Some sophisticated librarians (Voyetra) can handle several different types of synthesizers or even download rhythm patterns to electronic drum machines.

Patch editor programs differ from patch librarians in that they can actually alter the sound of a patch. Just as the term implies, MIDI system-specific messages usually are specific to a certain brand and type of synthesizer, so most patch editors are designed to work with only one or two types of machines. The system-specific messages that are sent by these edi-

If lightning still scares you, you're using the wrong file manager.



Be sure. Btrieve.®

Lightning may strike. But it doesn't have to destroy your database.

Btrieve® file management offers automatic file recovery after a system crash. So accidents and power failures don't turn into database disasters. Your Btrieve-based applications will come up when the lights come back on.

Fast. Btrieve is lightning fast, too. Its b-tree file structure automatically balances—you never waste time reorganizing the index. And Btrieve is written in Assembly language especially for the IBM PC. The result: electrifying speed on file maintenance routines. Applications that run fast. Users who don't waste time waiting.

The standard for networking. When your application requires multi-user file sharing, Btrieve (network version) sets the standard for the industry's most popular LANs: Novell, IBM, and 3COM, to name a few. Btrieve offers *safe* multi-user file management that coordinates simultaneous updates and protects against lost data.

Fully-relational data management. Using SoftCraft's entire family of products gives you a complete, fully-relational database management system. Xtrieve® speeds users through database queries with interactive, on-screen menus—no command language or special syntax. And you can add our report writer to Xtrieve to generate the reports you need.

For professional programmers. Btrieve is the fast, reliable answer for all your application development. In any development language—BASIC, Pascal, Cobol, C, Fortran and APL. With multikey access to records. Unlimited records per file. Duplicate, modifiable, and segmented keys. Variable length records.

With Btrieve, you can develop better applications faster. And know they'll be safe if lightning strikes.

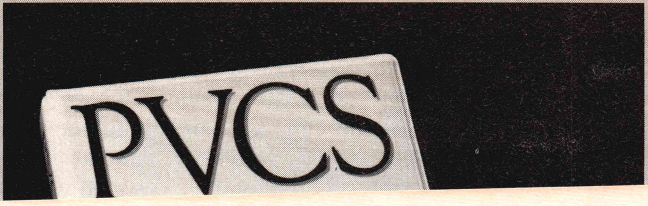
NO ROYALTIES

Suggested single/multi-user prices:
Btrieve: $245/$595
Xtrieve: $245/$595
Reporter: $145/$345
Requires PC-DOS or MS-DOS 2.X or 3.X.

Btrieve and Xtrieve are registered trademarks of SoftCraft, Inc.

SoftCraft Inc.
P.O. Box 9802 #917
Austin, Texas 78766
(512) 346-8380 Telex 358 200

CIRCLE 40 ON READER SERVICE CARD



PVCS

STOP

Fast, Free Product Info

Take this card with you as you read through this issue of Dr. Dobb's.

A reader service number appears on each advertisement. Circle the corresponding numbers at right for more info. Circle 999 to start a 12 month subscription at the price of $29.97

Name ____________________

Title ____________________

Company ____________________ Phone ____________________

Address ____________________

City/State/Zip ____________________

May 1987 #127 Expiration Date: August 31, 1987

Please circle one letter in each category:

I. My work is performed:
A. for in-house use only.
B. for other companies.
C. for end users/retailers.
D. in none of the above areas.

II. My primary job function:
A. Software Project Mgmt/Spvr
B. Hardware Project Mgmt/Spvr
C. Computer Consultant
D. Corporate Consultant
E. Other

III. My company department performs:
A. software development.
B. computer system integration.
C. computer manufacturing.
D. computer consulting.
E. computer research.
F. none of the above.

IV. This inquiry is for:
A. a purchase within 1 month.
B. a purchase within 1 to 6 months.
C. product information only.

V. Corporate Purchase Authority:
A. Final Decision-maker
B. Approve/Recommend
C. No Influence

VI. Personal Computer Users at my Jobsite:
A. 10,000 or more
B. 500 to 9,999
C. 100 to 499
D. 10 to 99
E. less than 10

VII. On average, I advise others about computers:
A. more than once per day.
B. once per day.
C. once per week.
D. less than once per week.

VIII. In my job function, I:
A. design software and/or write code.
B. design software.
C. write code.
D. don't design software or write code.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130
131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170
171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190
191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230
231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250
251	252	253	254	255	256	257	258	259	260
261	262	263	264	265	266	267	268	269	270
271	272	273	274	275	276	277	278	279	280
281	282	283	284	285	286	287	288	289	290
291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310
311	312	313	314	315	316	317	318	319	320
321	322	323	324	325	326	327	328	329	330
331	332	333	334	335	336	337	338	339	340
341	342	343	344	345	346	347	348	349	350
351	352	353	354	355	356	357	358	359	360
361	362	363	364	365	366	367	368	369	370
371	372	373	374	375	376	377	378	379	380
381	382	383	384	385	386	387	388	389	390
391	392	393	394	395	396	397	398	399	999

number, system version number, or specified date.

* Efficient disk storage. PVCS uses a very intelligent difference detection technique that minimizes the amount of disk space required to store a new version.

PVCS Maintains System Integrity

PVCS prevents system corruption that could ordinarily result from security breaks, user carelessness or malfunctions. The levels of security can be tailored to meet the needs of your project.

ibility of the Corporate PVCS, but excludes the features necessary for multiple-programmer projects. **$149**

Corporate PVCS — Offers additional features to maintain source code of very large and complex projects that may involve multiple programmers. Includes "Branching" to effectively maintain code when programs evolve on multiple paths (e.g., new versions for different systems, or a new program based on an existing program). Single user. **$395**

Network PVCS — Extends Corporate PVCS for use on Networks. File locking and security levels can be tailored for each project. 5-Station License $1,000. Call (503) 645-1150 for pricing on Licenses for more than 5 Stations. **$1000**

TO ORDER: VISA/MC **1-800-547-4000.** Dept. No. 325. Oregon and outside US call (503) 684-3000. Send Checks, P.O.s to: POLYTRON Corporation, 1815 NW 169th Place, Suite 2110, Dept. 325, Beaverton, OR 97006.

POLYTRON®

High Quality Software Since 1982

CIRCLE 283 ON READER SERVICE CARD

(continued from page 23)

tors are prefixed by an address code that tells other types of machines to ignore the data that follows, so patches can be sent to one device in a MIDI network without problems resulting from other machines misinterpreting the data. System-specific codes are usually published along with other technical data in a synthesizer's instruction manual.

Some vendors, such as Bacchus Software, specialize in patch editor programs. Bacchus' IBM PC-based editor is a SideKick-style memory-resident program that pops up over another running music program. Its main function is editing and archiving patches for the popular but difficult to program Yamaha DX-7 synthesizer. It is usually in the best interests of manufacturers to either write their own micro-based editors or to hire third-party software writers to support their products. For example, DigiDesign's waveform-editing programs allow data to be downloaded from sampling synthesizers, displayed, altered, and sent back to a sampler to be played. They support several brands of synthesizers, and manufacturers value their support because they make the samplers much more accessible and marketable. Some manufacturers even feature DigiDesign's software in their own ads and exhibits.

Sequencers

One of the many advantages MIDI affords is the ability to intercept and record musical events. The recorded data can be manipulated in ways that were impossible using standard audio tape recorders. Transposition (pitch shift), copying, and rearranging can be accomplished with software alone, and errors made during the recording process can be corrected without having to do retakes. Compositions can even be replayed while changing synthesizer patches, so a part that was originally written for a cello-type voicing could be tried out with a piano sound.

MIDI-based recorders or composition programs are sometimes known as sequencers (a carry-over from old analog instruments). In the modern context, a sequencer might be visualized as being much like an audio tape machine. Concepts such as multitracking, fast-forward, and rewind can be translated to software to ease the transition from conventional tape-oriented studios. New concepts such as quantization (automatic timing correction), real-time transposition, and complex looping constructs would be nearly impossible with the use of audio tape alone.

Sequencers are currently available for the IBM PC (Jim Miller, Voyetra), Macintosh (Southworth, Mark of the Unicorn), Amiga (Mimetics), Atari ST (Hybrid Arts), and Commodore-64 and Apple II (Doctor T, Passport). One of the original entries in the IBM field is Jim Miller's Personal Composer, which can record a performance in real time or let you enter data with a mouse or keyboard. Musical data can then be rearranged and edited using traditional staff-line notation and scores can be printed on common dot-matrix printers. Personal Composer includes a built-in patch editor for DX-7 synthesizers, a small graphics editor, and even a user-accessible LISP interface.

Personal Computer Interfaces

If you already own one of the computers mentioned, getting started in MIDI composition is as simple as purchasing the standard interface. You will then have access to a broad range of software packages that you can expand and update via disk (just like you do compilers and word processors). Some of the more common interfaces come from Passport (Commodore-64 and Apple II), Opcode (Macintosh), Mimetics (Amiga), and Roland (IBM PC and Apple II). The Atari ST series has a built-in MIDI interface. Most of these interfaces consist of little more than a UART for handling serial communication. The Roland MPU-401, however, includes an on-board microprocessor and timer for providing you with preprocessed, buffered data packets.

What MIDI Does (and Doesn't Do)

First of all, MIDI does not solve all existing problems in interfacing sound-generating equipment. For example, it is easy to send instructions for turning particular notes on and off, and sending a *NOTE-ON* command usually results in a predictable pitch from any two synthesizers. There is no standard for a violin sound or an oboe sound, however. Patches that are stored internally in the synthesizer can be requested via MIDI, but it is up to the individual programmer or manufacturer to devise the violin or oboe sound and assign a patch number to it. Manufacturers of lighting controls or mixing boards are on their own; system-specific commands tell other devices to ignore codes they won't understand, but there is no standard spec for how lights should respond to music commands.

Some parameters are standardized by MIDI—for example, voice messages include *NOTE-ON* and *NOTE-OFF* events. Notes are assigned numbers from 0–7fh and simply turned on and off. Velocity information is included with the command and can be interpreted as loudness. Channel addresses designate a particular oscillator or synthesizer that will respond to the command—bytes from 90h to 9fh turn on notes for channels 0–0fh; bytes from 80h to 8fh turn the same notes off. For example, the 3-byte command to turn on note number 32h, on channel 3, at velocity 22h is *93h, 32h, 22h*; the command to turn off the same note with a turn-off velocity of 10h is *83h, 32h, 10h*. Velocity is usually not relevant when turning a note off, so sometimes a *NOTE-ON* command with a velocity of zero is used as a *NOTE-OFF*. Other voice commands include key pressure and pitch bend, but these are not sent as part of the *NOTE-ON* or *NOTE-OFF* message packets.

MODE MESSAGES control the response characteristics of the receiving device. Synthesizers can be told to listen to a specific channel (*OMNI-OFF*) or to respond to messages on all channels (*OMNI-ON*). In addition, provision is made to address one oscillator per channel (*MONO MODE*) or to allow the synthesizer's internal software to assign its own voices so messages can be sent over one channel (*POLY MODE*). Combinations of these yield four modes.

SYSTEM-REAL-TIME and *CLOCK MESSAGES* allow synchronization of all machines in the chain by sending a periodic software clock tick over the MIDI bus.

STOP

Take this card with you as you read through this issue of Dr. Dobb's.

Fast, Free Product Info

A reader service number appears on each advertisement. Circle the corresponding numbers at right for more info. Circle 999 to start a 12 month subscription at the price of $29.97

Name ____________________________

Title ____________________________

Company ________________ Phone ____________

Address ____________________________

City/State/Zip ____________________________

May 1987 #127 Expiration Date: August 31, 1987

Please circle one letter in each category:

I. My work is performed:
A. for in-house use only.
B. for other companies.
C. for end users/retailers.
D. in none of the above areas.

II. My primary job function:
A. Software Project Mgmt/Spvr
B. Hardware Project Mgmt/Spvr
C. Computer Consultant
D. Corporate Consultant
E. Other

III. My company department performs:
A. software development.
B. computer system integration.
C. computer manufacturing.
D. computer consulting.
E. computer research.
F. none of the above.

IV. This inquiry is for:
A. a purchase within 1 month.
B. a purchase within 1 to 6 months.
C. product information only.

V. Corporate Purchase Authority:
A. Final Decision-maker
B. Approve/Recommend
C. No Influence

VI. Personal Computer Users at my Jobsite:
A. 10,000 or more
B. 500 to 9,999
C. 100 to 499
D. 10 to 99
E. less than 10

VII. On average, I advise others about computers:
A. more than once per day.
B. once per day.
C. once per week.
D. less than once per week.

VIII. In my job function, I:
A. design software and/or write code.
B. design software.
C. write code.
D. don't design software or write code.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130
131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170
171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190
191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230
231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250
251	252	253	254	255	256	257	258	259	260
261	262	263	264	265	266	267	268	269	270
271	272	273	274	275	276	277	278	279	280
281	282	283	284	285	286	287	288	289	290
291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310
311	312	313	314	315	316	317	318	319	320
321	322	323	324	325	326	327	328	329	330
331	332	333	334	335	336	337	338	339	340
341	342	343	344	345	346	347	348	349	350
351	352	353	354	355	356	357	358	359	360
361	362	363	364	365	366	367	368	369	370
371	372	373	374	375	376	377	378	379	380
381	382	383	384	385	386	387	388	389	390
391	392	393	394	395	396	397	398	399	999

Fast, Free Product Info

STOP

Take this card with you as you read through this issue of Dr. Dobb's.

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #217, CLINTON, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. Box 2157
Clinton, Iowa 52735-2157

Clarify and document your source listing and get an "organization chart" of your program's structure

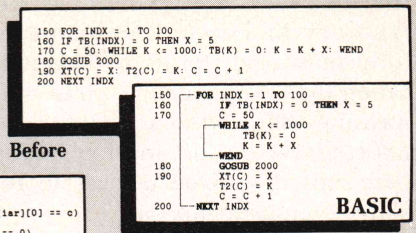
with two NEW utilities from Aldebaran Laboratories, for C, BASIC, d-BASE® Pascal and Modula-2 programmers.

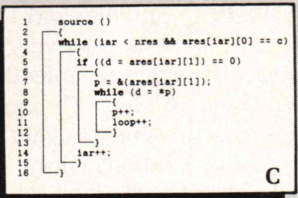
"Occasionally, a utility comes along that makes a programmer's life much easier. SOURCE PRINT is such a program. It contributes to the programmer's job by organizing code into a legible format and by helping to organize the documentation and debugging process."

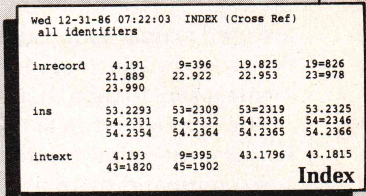
— PC Magazine Sept. 16, 1986

Source Print™

organizes your source code, simplifies debugging, and makes documentation a snap! It lists one or more source files with informative page headings and optional line numbers, while offering invaluable features:

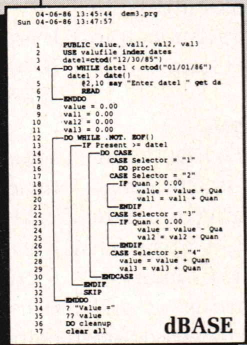






The Index (Cross-Reference list) saves you time by showing exactly where variables are used and where functions, procedures, and routines are called. Locations where new values may be assigned to variables are shown, making it easy to track down that mysterious value change.

$75.00



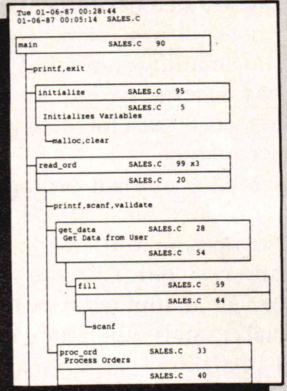
Structure Outlining solves the problem of hard-to-see nested control structures by automatically drawing lines around them.

Automatic Indentation of source code and listings reduces your editing time and ensures indentation accuracy.

Plus . . . Source Print generates a table of contents listing functions and procedures. Keywords can be printed in boldface on most printers. Multi-statement BASIC lines can be split for readability. Functions and procedures can be drawn by name from one or more source files to form a new file.

Tree Diagrammer™

shows your program's overall organization at a glance. Ordinary program listings merely display functions, procedures, and subroutines sequentially, but do not display the relationships between these routines. Our revolutionary new Tree Diagrammer automatically creates an "organization chart" of your program showing the hierarchy of calls to functions, procedures, and subroutines. Recursive calls are indicated and designated comments in the source code will appear on the chart.



Tree Diagrammer helps you organize your program more logically. And you'll be amazed at how easy it is to debug when you see how your routines interact.

$55.00

Source Print and Tree Diagrammer both have easy-to-use menus with point-and-shoot file selection, and let you search for files containing a given string. For IBM PC and compatibles with 256K.

Join thousands of programmers who are working more efficiently using Source Print and Tree Diagrammer. Order these indispensable tools today. We ship immediately, and there's no risk with our 60-day money-back guarantee. **Order both and save. Only $125.00.**

800-257-5773 Dept. 55
In California:
800-257-5774 Dept. 55

MasterCard, VISA, American Express, COD. Add $5 for shipping/handling.

Source Print and Tree Diagrammer are trademarks of Aldebaran Labs. dBASE is a trademark of Ashton Tate. Prices subject to change without notice.

Aldebaran Laboratories 3339 Vincent Rd. Pleasant Hill, CA 94523 415-930-8966

YES! Rush me ☐ Source Print @ $75. ______ ☐ Tree Diagrammer @ $55. ______
☐ Both $125. Ship/Handling $5. For CA add 6% tax ______ Total ______
Name ______
Company ______
Address ______
City ______ State ______ Zip ______
☐ Check enclosed ☐ VISA ☐ MasterCard ☐ American Express
Card # ______ Exp. Date ______
Signature ______ Phone # ______

(continued from page 26)

SYSTEM-SPECIFIC commands address synthesizers from a specific manufacturer. These commands are used to send patches and other data that has meaning only to certain types of synthesizers. Each manufacturer must apply for its own identification byte (Sequential Circuits = 1, Kawai = 40h, and so on). An *END OF EXCLUSIVE* (*EOX*) or another status byte tells deselected devices to resume listening to bux data.

SYSTEM COMMON messages address all synthesizers on line. They are used primarily for setup information, such as selecting songs or telling synthesizers to tune their oscillators.

The first byte of all MIDI commands has its most-significant bit (MSB) set to 1, so command bytes are always between 80h and 0ffh. Data bytes have their MSBs reset (0), so their range is 0 to 7fh. If a given block of data contains values greater than 7fh, all bytes in the block are broken into 7-bit nibbles and sent in two parts. This keeps the MSBs reset so that receivers can always stay in sync—even if a byte is lost.

MIDI Hardware Specification

The original MIDI specification made some trade-offs between speed, economy, and efficiency, which inevitably resulted in performance compromises. It is easy to point out shortcomings now that the interface has become widely known, but the low cost had a lot to do with its initial acceptance.

MIDI is a serial protocol that communicates on a 31.25-kHz, optically isolated current loop. The odd baud rate resulted from the reluctance of earlier manufacturers to install special crystals when they usually had a 1–4-MHz processor clock available (31.25 kHz=1 MHz/32). Use of a convenient single-chip binary divider yields the 31.25-kHz UART clock.

Optical isolation is required for eliminating ground loops and isolating sensitive audio equipment from the high frequencies in computer gear. An optically isolated parallel interface could have been specified, but serial interfacing decreases the cost for the isolators and simplifies cabling. The penalty, of course, is speed. Clock rates are limited by cable capacitance and by response times for economical optos and UARTs. Some manufacturers are now using 62.5-kHz (double frequency) rates for downloading waveform samples or other data-intensive applications, but it is unlikely that the baud rate standard will change in the near future.

The optos at the receiving end of each MIDI link require about 5 mA to turn on. Because the loop sends a current (like old Teletype machines do), it is relatively immune to noise as long as cables don't extend more than 50 feet. The built-in current limit resistors prevent star-network configurations (only one receiver can be hooked to a transmitter), so a third port (the MIDI THRU port) is included on most synthesizers to allow daisy-chaining of receivers. The MIDI THRU port duplicates the data coming from the MIDI IN port and retransmits it to the next machine in the chain.

Speed Considerations

Most MIDI *NOTE-ON* or *NOTE-OFF* commands require 3 bytes at 320 microseconds per byte, so turning on a note on the synthesizer takes approximately 1 millisecond. The human ear is more sensitive to starting (attack) transients than to ending (decay) timings—for psychoacoustic reasons and simply because the played notes usually start off sharply and taper off before their final decay. This means that even in best-case circumstances when no other events are being sent over the MIDI bus, starting transients for ten *NOTE-ON* events will be spread apart by 10 milliseconds. This approaches the threshold of audible delay, and additional notes may have a slap-echo effect. Similarly, large chords may be audibly arpeggiated.

To avoid objectionable delays, some MIDI hardware now features multiple MIDI OUT ports. The computer sends parallel commands to each port to avoid daisy-chaining delays. These should not be confused with MIDI THRU ports, which track the MIDI IN port. Multiple MIDI IN ports are less common but certainly helpful when recording events coming from more than one source. Because MIDI is a multibyte protocol, merging and recording two sources can be fairly complex if only one input port is available.

Designing a Sequencer

MIDI control software can be written in any language, but fast queuing of serial data is important. The majority of software writers I know use a mixture of C and assembly language. It may be convenient to use a compiler such as Wizard C, which allows dropping into assembly language for I/O access or speed.

Small computers can be used, but be aware that RAM can be used up quickly. If storage for a single note uses 8 bytes, an eight-finger chord will use 64 bytes, and playing eight of these chords in one measure will use 0.5K RAM. Linear address space is easy to allocate and control, and segmented architectures present no large problems because a segment is usually more than enough to record any single sequence of note events (a *track* in recording terminology). Bank-select RAM is usually a problem when several tracks are played back in parallel. If the tracks are stored in separate banks, the switching overhead may be cumbersome.

Most of my current designs use IBM PCs and ATs, so some of the examples focus on 8088 designs, but the design principles will adapt easily to other computers.

Designing Hardware

Custom hardware affords some measure of software security and may provide functions not available on existing interfaces. Because some designers prefer using their own hardware, I will provide a few guidelines.

MIDI's serial protocol requires no hardware handshake signals, so 40-pin UARTs are not necessary. The only small UART that may cause trouble is the Intel 8251. I have used Motorola 68B50s in several designs with good results. On 8088 systems, Motorola's E (enable) signal can be developed by ANDing the port read and write signals together.

Timers get tricky when multiple devices are hooked to one interrupt line. I have used Intel 8253s, but I usually connect the output line to a flip-flop so that the output pulse can be trapped and identified. Flip-flops are not necessary if the timer interrupt is isolated because the 8259 interrupt controller has edge-triggering.

Most MIDI programs that run on the IBM PC use Roland's MPU-401 interface. Timers are not required on IBM PCs that use this interface, but if you are designing your own, try to include an on-board timer. The PC's internal timers do not provide the accuracy necessary to deal with high-resolution music timing. For low-resolution applications, IRQ 0 from the PC motherboard can be readjusted to run at a multiple (*X*) of its normal speed. Then, every *X* pulses, the old interrupt service routine is run. The interrupt acknowledge should be skipped when jumping to the old routine. Remember to reset the interrupt vectors before exiting to DOS. Disk head loads use timer 0 for timeouts, so problems with this routine usually cause the drive light to stay on.

Slower clock rates for hardware timers obviously result in decreased resolution. Not so obvious, though, is the way that tempo resolution and note-timing accuracy combine to make even tougher demands on the system timer's crystal frequency. I try to clock the timer at around 2 MHz so that I can record with a resolution of about 96 divisions per quarter note while maintaining sufficient accuracy in specifying tempos.

In most applications, the divisor sent to the hardware timer is used to trim the tempo, which is set in increments of beats-per-minute (BPM). An easy way to control tempo is to index into a table of divisors by the desired number of BPMs. I normally generate the tables with a C program that does the calculations and prints out the tables exactly as they should appear in the sequencer program. When the values are verified, I redirect the output of the program into a file that is then compiled.

Interrupts

The interrupt service routine (ISR) is responsible for prioritizing interrupts and coordinating all incoming data. Obscure problems with the ISR can propagate through the entire system. A flowchart will probably help to clarify possible timing errors or bottlenecks before the ISR is coded.

It might appear that a system using only serial ports would pose no problems in I/O handling, but MIDI baud rates are high, and there may be multiple ports. Input interrupts should be serviced as quickly as possible because losing a byte may result in losing an important *NOTE-OFF* message. When the recorded events are retransmitted on playback, the synthesizer will play a stuck note until the operator can figure out a way to generate a *NOTE-OFF* (sometimes leaving an embarrassed performer desperately groping for the power switch). Output interrupts are less critical; a momentary delay is the only penalty for slow output response times. Be particularly careful how these two interrupt sources are handled when UART hardware lines are shared.

On the IBM PC, the first instruction in the ISR can be an STI (the BIOS does this) for reenabling higher-priority interrupts. Lower-priority interrupts can be enabled by masking the present interrupt and sending an acknowledge (EOI) to the interrupt controller. The pending interrupt cannot be retriggered until the source of the interrupt is reset (UART is read and so on). Make sure that the pending interrupt line is high (active) when sending the EOI because obscure problems can result with the 8259 when it cannot find the source

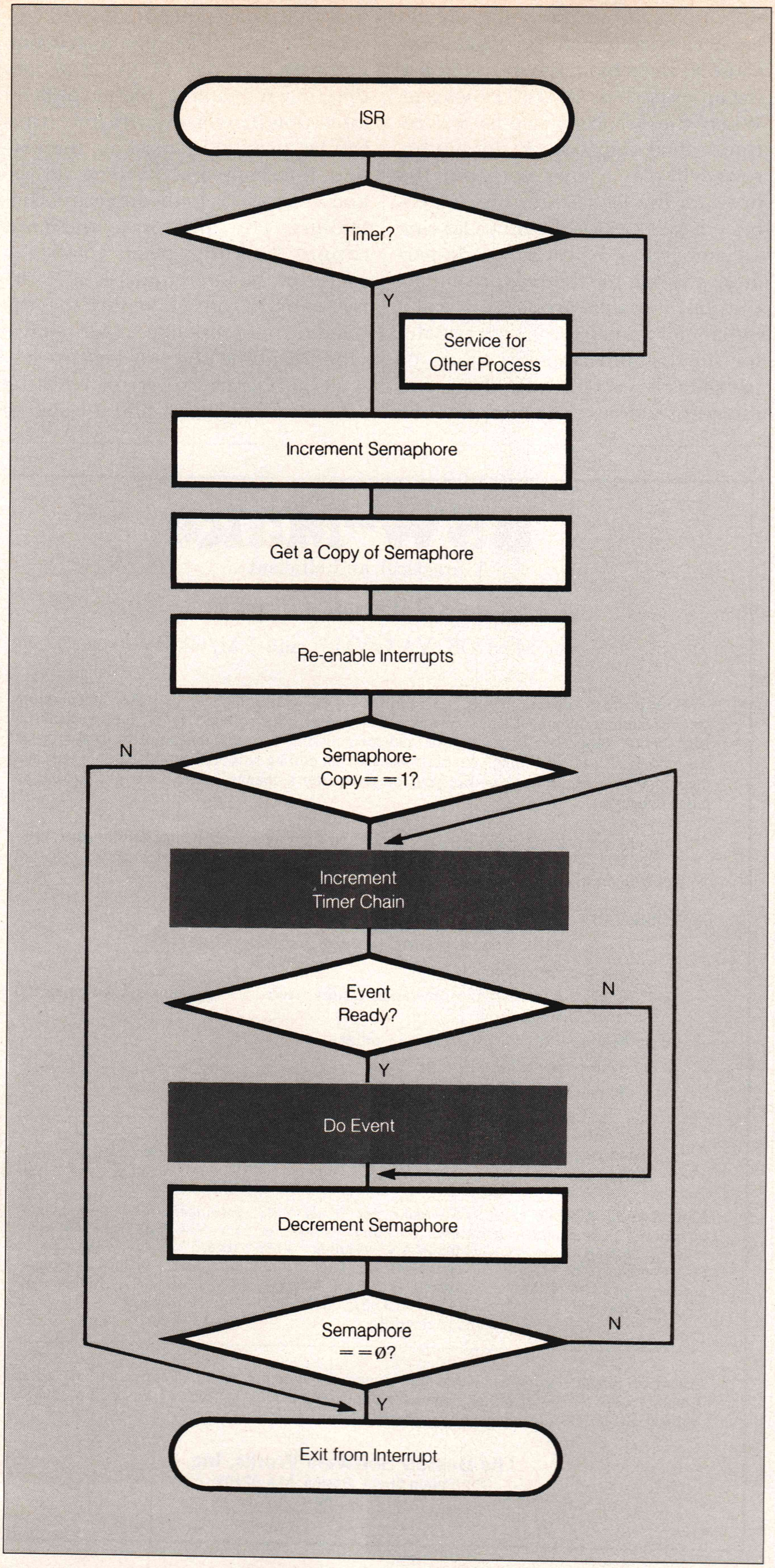


Figure 1: *Flowchart for timer interrupt using semaphore*

MUSIC RECORDER
(continued from page 29)

of the interrupt being acknowledged. It may also help to poll the 8259 registers to make sure that the interrupt line is low before exiting from the ISR. This will help to avoid difficulty with the PC's edge-triggered hardware.

Timer interrupts are important, but they can generally be considered a lower priority than UART interrupts. The timer routine itself is usually short, consisting of little more than incrementing a series of software counters, but at some point compares must be made with target values and a long series of events may be triggered.

The timer interrupt conveys no actual data aside from a flag, so normal queues are not necessary. The best way to enqueue timer interrupts is with a semaphore, which allows the ISR itself to be interrupted. When the timer tick occurs, the semaphore is incremented and timer interrupts are re-enabled. If the semaphore has been incremented to 1, no interrupts are nested and normal processing can resume. If the value is greater than 1, this means that the timer interrupts have stacked up, so the interrupt is exited and control is returned to the timer ISR that was interrupted. Before the timer ISR exits, the semaphore is decremented, and if the value is still nonzero, the interrupts must have been nested. In this case, control is returned to the top of the timer ISR, which loops until the semaphore returns to zero. This allows the ISR to catch up with lost interrupts without losing any timer pulses or UART interrupts.

The flowchart in Figure 1, left, is simplified; it may help to refer to the code listing (Example 1, page 31) for more subtle details. Make sure the semaphore is initialized to zero at start-up or the timer ISR will never run.

Timing Notes

The MIDI sequencer usually uses a hardware clock for its main timing reference, but it may be necessary to synchronize to an external software clock provided by a drum machine or timing converter. I will refer to these as internal and external sync,

respectively. MIDI software clocks occur at a standard rate of 24 per quarter note, which provides musical resolution to within a 64th note triplet. This sounds as though it would keep up with even the fastest musicians, but remember that you are dealing with timing edges. Trimming these edges to the nearest 24th of a quarter note may cause the recorded notes to sound too symmetrical, or mechanical

Conversely, it may seem that slowing the system clock could correct the timing of inaccurate note values. This rounding (quantization) is sometimes used to advantage, but overuse removes the human signature and creates a metronomic, mechanical sound. Uneven qualities are most often missed on parts such as solos, which appear up front in a composition. Obviously, all devices synchronized with MIDI real-time clock signals will be quantized to some extent.

Time Stamps

In order to maintain precise timing of events while allowing interrupts to proceed at full speed, I store a time stamp with each event coming into the UART receive queue. Very simply, if a received byte is greater than 7fh (MSB is set on all commands), the current time is enqueued after the byte. This provides freeze-frame timing; the time record travels with the received data until the program is ready to process it. Only the leading byte needs to be time-stamped. Follow-up data (with MSBs reset) are assumed to have been received at the same time. This technique can provide accuracy better than that obtainable by waiting for the complete record and processing it instantly. Usually the sending device intends that all bytes be received simultaneously, so stamping the leading byte will more accurately reflect the actual event timing, even if the transmitter lags in sending the follow-up data.

Real Time

The MIDI specification calls for two basic types of software clocks. Clock-in-stop (0fch) allows receivers to phase-lock to the clock frequency prior to start-up. When the transmitter switches to clock-in-play (0f8h), all synchronized receivers switch to their active state (usually playback or

```
;
; TIMER_ISR
; This partial listing will help to model a timer int. based on semaphores
;
dseg    segment para public 'dseg'
 t_sem  dw      0               ; make sure this value always starts at
                                                                   zero
dseg    ends

cseg    segment para public 'cseg'
        assume cs:cseg, ds:dseg

        public  int_vector
int_vector proc far
        sti
        push    si              ; save all registers
        push    di
        push    ax
        push    bx
        push    cx
        push    dx
        push    ds
        push    es
        ;

        mov     di,dseg         ; set up for access to data seg in this
                                                                 module
        mov     ds,di
 again:                         ; loop point for retries if more ints hit
        mov     si,[t_sem]      ; 'sample' the semaphore before increment !
        inc     [t_sem]
        ;
        mov     al,20h          ; this will acknowledge interrupts on IBM
                                                                     PC
        out     20h,al
        ;
        call    reset_timer_ff  ; toggle flip-flop hooked to the timer chip
        ;
        ; Interrupts are now re-enabled
        ;
        or      si,si           ; Check the 'sampled' semaphore
        je      i_loop          ; If semaphore was zero, execute tmr
                                                        routine
        ;
        ; Skip the main timer routine if this is a nested int - check for
                                                                   ovfl
        ;
        cmp     si,200          ; Too many interrupts stacked-up?
                                                    (overflow?)
        jc      skip            ; If not, exit directly.
        ;
        ; Interrupts have overflowed -- stop and check timer values
        ;
        call    emergency_stop  ; semaphore overflow! timers are set too
                                                                   fast
        jmp     skip
        ;
i_loop:
        ;
        ; Execute the main timer routine - then check for stacked interrupts
        ;
        call    timer_routine   ; run main timer chain
        ;
        dec     [t_sem]         ; if sem dec's to zero, no ints are nested
        jnz     i_loop          ; if ints ARE nested, loop back to catch up
skip:
        pop     es
        pop     ds
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     di
        pop     si              ; restore registers and return
        iret
int_vector endp
cseg    ends
```

Example 1: *A MIDI interrupt service routine with a semaphore*

(continued from page 31)

record). To maintain accuracy, real-time messages are transmitted at any time—even in the middle of other multibyte messages. Receivers must account for this possibility even if the real-time messages are not used. Both clocks are always sent at the rate of 24 per quarter note. Altering the clock frequency changes tempo, not accuracy.

Other real-time messages include *START-FROM-BEGINNING* (0fah), which resets internal song pointers; *CONTINUE* (0fbh), which tells receivers to resume from the current location; and *ACTIVE SENSING* (0feh), which just lets the receivers know that the transmitter is still there. The latter is optional and used notably by the Yamaha DX-7 synthesizer. When using a DX-7, you will probably want to discard the 0feh bytes because they will be received constantly—even when not recording. They can fill up the input queues if the input interrupts are enabled.

The last real-time message, *SYSTEM RESET* (0ffh), is dangerous because it could start a regenerating condition in which every component in the system sends resets to each other. It is usually reserved for linkage to a hardware reset switch or used judiciously by the master controller.

Storage Formats

There is no standard yet for either RAM- or disk-based storage for MIDI events. I have heard rumors of a standard for disk storage that would allow one manufacturer's software to read files written by someone else, but intermediate RAM storage is another story. The storage formats used throughout the industry are diverse and usually so complex that changing internal formats would require extensive rewrites.

Most internal storage methods fall into one of four categories that I call end-point-relative, end-point-absolute, single-point-absolute, and bar-and-note storage. All storage formats involve storing data in a linear data stream. Relative timing implies that timing is encoded as a distance from the previous event. Absolute timing uses a global time reference, such as beats and bars. End-point storage refers to separate storage locations for *NOTE-ON* and *NOTE-OFF* events (usually with their own time stamps). Single-point storage requires that a pointer be aimed at the *NOTE-ON* record in the data stream, and when the *NOTE-OFF* event is received, it is stored at the same location (better yet, the note duration can be computed and stored). The bar-and-note method parallels the way music is normally notated. Each method has advantages, and there is a lot of overlap between categories.

I try to carry MIDI's philosophy of setting MSBs of leading bytes when encoding data for storage in the stream. This allows resyncing if a byte is missed and makes data streams easier to edit. Many software writers use this method for internal storage.

End-Point-Relative Storage

MIDI data is received as a stream of bytes with high bits (MSBs) set on commands and reset on data. Why not store the bytes just as they are received? Embedded MIDI clock messages provide the proper spacing for *NOTE-ON*, *NOTE-OFF*, and other events. Replaying the data requires setting up a series of play pointers into the data stream (one for each track to be played). When the start command is received, the data is sent to the output UARTs just as it was received, waiting for a 24th of a beat every time a clock command is encountered in the stream. Unfortunately, this method uses RAM storage even if no events are being transmitted, and multiple channels store multiple copies of all unnecessary timing bytes.

In Example 2, below, it is assumed that an external MIDI clock provides the timing. Even if an internal timer is used, 0f8h bytes can be inserted into the input queue to simulate the MIDI software clock. A refinement of this method conserves storage by counting all received clock bytes. When an event is received, the accumulated count is stored before the event and then the counter is reset.

End-Point-Absolute Storage

Time stamping requires a system timer that is incremented every time a MIDI clock or timer interrupt is received. When MIDI data is enqueued, the system timer is copied into the queue as the time stamp. When the data is dequeued for storage, the time stamp is stored with the data record to provide an accurate absolute timing reference. Unlike relative timing, this allows you to locate any spot in the data stream without counting all embedded timing bytes. To replay time-stamped data, restart the system timer and wait until it matches the timing bytes of the first item in the data stream. Then send the data (without the time stamp), and advance the stream pointers.

Example 3, page 37, uses only 2 bytes for the time stamp and for the system timer. More practical systems use 3 bytes to allow more than two hours' recording time before the timer overflows (at 96 pulses per quarter). The low-order byte is incremented until it reaches 96 (or 24 for MIDI clock timing). Then the byte is reset and the count propagates through the other two timer bytes. The top timer bytes turn over at 127, so the MSBs are always zero.

Single-Point-Absolute Storage

Single-point storage requires maintaining a list of pointers to access active notes in the data stream. When a *NOTE-ON* is received, it is enqueued and stored in the data stream in the

```
- f8h --------- f8h 94h 46h 32h -------------- f8h 84h 46h 16h ---------- f8h
|wait 24th     |wait 24th, then               |wait 24th, then
|              |    output a NOTE-ON on ch.4  |    output a NOTE-OFF on ch.4
|              |        for note no. 46h      |        for note no. 46h
|              |          velocity = 32h      |          OFF velocity =16h
```

Example 2: *End-point-relative storage*

PHANTASTIC!
A SUPER SALE ON
PHOENIX PRODUCTS

Plink®86*plus*

Plink86plus lets you go beyond your personal computer's 640K memory limitation. This highly advanced DOS overlay linker provides capabilities that are unmatched by any other DOS linker.

Large, Efficient Programs

With Plink86plus, you can create very large programs that use very little memory — programs as large as 16 megabytes run in as little as 192K! And advanced memory management features ensure that even the largest programs will execute efficiently.

Automatic Overlays

The automatic overlay technique of Plink86plus lets you get your programs up and running fast. You can divide your program into as many as 4095 tree-structured overlay areas and change its structure anytime without modifying the program source code. You can store the overlays in separate files, separate directories or on separate disks. You have complete flexibility.

Wide Support

Plink86plus supports any compiler or assembler producing standard Intel or Microsoft OBJ files. It works with all popular language systems including FORTRAN, COBOL, Lattice C, CI C-86 and any Microsoft or IBM language.

Pfix®86*plus*

Pfix-86plus is an easy-to-use, menu-driven, symbolic debugger that works with any Microsoft or IBM compiled language.

Symbolic Debugging

Pfix86plus automatically handles Plink86plus overlaid or resident programs. It can also access the full symbol table provided by MS Link. Source code, assembly language translations, stack, data areas and breakpoints all appear simultaneously on the multiwindow display.

Full-featured

Pfix86plus features include an in-line assembler, breakpoint settings, full speed or trace modes, user-assignable variables, dual-monitor support, traceback, debug log, synchronized source code display, breakpoints in source code, disassembly to disk and much more.

PforCe™

PforCe is a comprehensive library of over 400 easy-to-use C language functions designed to significantly reduce your program development time.

Complete C Library

PforCe provides high-level subsystems for B-tree databases, windows, field editing, screen editing, time/date calculations, directory management, error-trapping, etc. It also provides low level functions that can interface directly to your hardware. It has interrupt-driven communications, string handling, menus, keyboard background tasking and complete DOS control. Other features include disk management, archiving, compilation, library management utilities and much more. PforCe comes with full library source code and demo programs.

And for C++ Users

PforCe++ is an alternate version of PforCe that's specifically designed for object-oriented programming with C++.

Pfinish™

Pfinish is a high performance execution profiler that helps you locate inefficiencies in programs which slow execution or waste memory space.

Execution Snapshot

Pfinish provides you with a complete statistical snapshot of your program in execution. Its comprehensive output reports may be displayed or printed in either tabular form or histograms. The results can be sorted in many ways. By examining these reports, you can determine the number of times various sections of code in your program have been executed.

Pinpoint Accuracy

Pfinish lets you pinpoint execution down to the exact subroutine line number. By inserting markers inside executable programs, you can track the number of times your program passes each marker and determine which other routines were affected by individual calls.

Overlay Support

Pfinish fully supports overlaid programs created with Plink86plus.

Pfantasy Pac™

Phoenix combination package consisting of Plink86plus, Pfix86plus, Pfinish, Pmaker, Pmate and Ptel.

CIRCLE 98 ON READER SERVICE CARD

Make the Connection . . .

Phoenix Technologies is well respected in the world of personal computer programming. We at Programmer's Connection are pleased to be your complete source for these high quality software products.

phoenix products

		List	Ours
Pasm86 Macro Assembler Version 2.0		195	**108**
Pdisk Hard Disk & Backup Utility		145	**89**
Pfantasy Pac Phoenix Combo	*Sale*	1295	**799**
Pfinish Execution Profiler	*Sale*	395	**209**
Pfix86plus Symbolic Debugger	*Sale*	395	**209**
PforCe Comprehensive C Library	*Sale*	395	**209**
PforCe++ Library for Guidelines C++	*Sale*	395	**209**
Plink86plus Overlay Linker	*Sale*	495	**279**
Pmaker Make Utility		125	**78**
Pmate Macro Text Editor		195	**108**
Pre-C Lint Utility		295	**154**
Ptel Binary File Transfer Program		195	**108**

Prices include **FREE** UPS surface shipping to all U.S. customers. Express services are also available at the shipping carrier's standard rate.

Please refer to our main advertisement in this journal for ordering information and the largest advertised selection of programmer's development tools for IBM personal computers and compatibles.

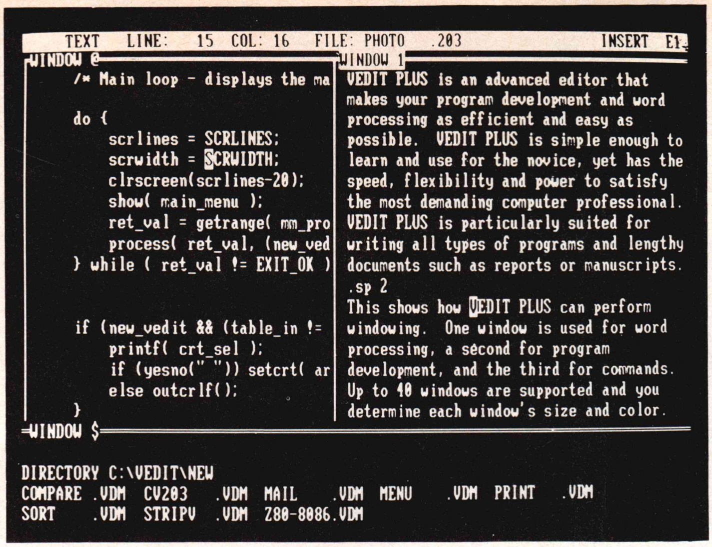
CALL TOLL FREE

U.S. 800-336-1166
CANADA 800-225-1166
OHIO and ALASKA 216-877-3781
(Call Collect)

OVERSEAS 216-877-3781
CUSTOMER SERVICE 216-877-1110

PROGRAMMER'S
QUALITY INTEGRITY
Service
SUPPORT PRICE
CONNECTION

Programmer's Connection
136 Sunnyside Street
Hartville, Ohio 44632



VEDIT PLUS

#1 PROGRAMMABLE EDITOR

(Call for FREE DEMO disk)

Stunning speed. Unmatched performance. Total flexibility. Simple and intuitive operation. The newest VEDIT PLUS defies comparison.

Try A Dazzling Demo Yourself.

The free demo disk is fully functional - you can try all features yourself. Best, the demo includes a dazzling menu-driven tutorial - you experiment in one window while another gives instructions.

The powerful macro programming language helps you eliminate repetitive editing tasks. The impressive demo/tutorial is written entirely as a 'macro' - it shows that no other editor's 'macro' language even comes close.

Go ahead. Call for your free demo today. You'll see why VEDIT PLUS has been the #1 choice of programmers, writers and engineers since 1980.

Available for IBM PC, TI Professional, Tandy 2000, DEC Rainbow, Wang PC, MS-DOS, CP/M-86 and CP/M-80. (Yes! We support windows on most CRT terminals, including CRT's connected to an IBM PC.) Order direct or from your dealer. $185.

Compare features and speed

	BRIEF	Norton Editor	PMATE	VEDIT PLUS
'Off the cuff' macros	No	No	Yes	Yes
Built-in macros	Yes	No	Yes	Yes
Keystroke macros	Only 1	No	No	100 +
Multiple file editing	20 +	2	No	20 +
Windows	20 +	2	No	20 +
Macro execution window	No	No	No	Yes
Trace & Breakpoint macros	No	No	Yes	Yes
Execute DOS commands	Yes	Yes	Yes	Yes
Configurable keyboard Layout	Hard	No	Hard	Easy
'Cut and paste' buffers	1	1	1	36
Undo line changes	Yes	No	No	Yes
Paragraph justification	No	No	No	Yes
On-line calculator	No	No	No	Yes
Manual size / index	250/No	42/No	469/Yes	380/Yes
Benchmarks in 120K File:				
2000 replacements	1:15 min.	34 sec.	1:07 min.	6 sec.
Pattern matching search	20 sec.	Cannot	Cannot	2 sec.
Pattern matching replace	2:40 min.	Cannot	Cannot	11 sec.

VEDIT PLUS FEATURES

- Simultaneousy edit up to 37 files of unlimited size.
- Split the screen into variable sized windows.
- 'Virtual' disk buffering simplifies editing of large files.
- Memory management supports up to 640K.
- Execute DOS commands or other programs.
- MS-DOS pathname support.
- Horizontal scrolling - edit long lines.
- Flexible 'cut and paste' with 36 'scratch-pad' buffers.
- Customization - determine your own keyboard layout, create your own editing functions, support any screen size.
- Optimized for IBM PC/XT/AT. Color windows. 43 line EGA.

EASY TO USE

- Interactive on-line help is user changeable and expandable.
- On-line integer calculator (also algebraic expressions).
- Single key search and global or selective replace.
- Pop-up menus for easy access to many editing functions.
- Keystroke macros speed editing, 'hot keys' for menu functions.

FOR PROGRAMMERS

- Automatic Indent/Undent for 'C', PL/I, PASCAL, etc.
- Match/check nested parentheses, i.e. '{' and '}' for 'C'.
- Automatic conversion to upper case for assembly language labels, opcodes, operands with comments unchanged.
- Optional 8080 to 8086 source code translator.

FOR WRITERS

- Word Wrap and paragraph formatting at adjustable margins.
- Right margin justification.
- Support foreign, graphic and special characters.
- Convert to/from WordStar and mainframe files.
- Print any portion of file; selectable printer margins.

MACRO PROGRAMMING LANGUAGE

- 'If-then-else', looping, testing, branching, user prompts, keyboard input, 17 bit algebraic expressions, variables.
- Flexible windowing - forms entry, select size, color, etc.
- Simplifies complex text processing, formatting, conversions and translations.
- Complete TECO capability.
- Free macros: • Full screen file compare/merge • Sort mailing lists • Print Formatter • Menu-driven tutorial

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103 (313) 996-1299, TELEX 701821

CIRCLE 122 ON READER SERVICE CARD

SUBSCRIBE & SAVE

$5 SAVINGS

Subscribe to DR. DOBB'S JOURNAL and save over $5—a 15% savings off the cover price!

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express
☐ Payment enclosed ☐ Bill me later

Card # ______________________ Exp. date __________

Signature ______________________

Name ______________________

Address ______________________

City ______________ State ______ Zip ______

ONLY $29.97! YOU SAVE OVER $5.00

Savings based on a full one-year cover price of $35.40. Canada and Mexico add $10 for surface mail per year. All countries add $27 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. 3446

COMMENTS & SUGGESTIONS

May 1987, #127

Dear Reader,

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take the time to write a letter. This card provides you with a quick and easy way to correspond and, if you include your name and address, we may use appropriate comments in The Letters column. Simply fill it out and drop it in the mail.

—Ed

Which articles or departments did you enjoy the most this month? Why?

____

Comments or suggestions: ______________________

Name: ______________________

Address: ______________________

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

No Postage Necessary If Mailed In The United States

Dr. Dobb's Journal of Software Tools
Box 3713
Escondidio, CA 92025-9843

(continued from page 32)

same way as in end-point-absolute storage, but zeros are stored afterward in a compartment reserved to keep track of the note's duration. A pointer to the zero bytes is stored in a list to allow access to the duration. Every time a MIDI clock byte or timer interrupt is received, the list is checked and the pointers are used to increment duration bytes. When the *NOTE-OFF* event is received, the pointer is removed from the list—no other action is necessary. The duration will be frozen as part of the note record. Single-point-absolute storage provides advantages in editing (notes can be moved easily) and display (it is easy to tell whether a note is a quarter note, half note, and so on).

In Example 4, below, during playback *NOTE-ON*s are derived in the usual way (wait until the system timer reaches the embedded time stamp), but this time the duration bytes are retrieved and stored in a time-out list. They are decremented with every timer tick, and when they reach zero, a *NOTE-OFF* is transmitted. The time-out list must keep a copy of the note number so that the proper note can be turned off. The single-point method affords another advantage: it is unlikely that notes will get stuck. *NOTE-OFF* events cannot be missed. Any reasonable value in the time-out list will eventually decrement to zero and cause a *NOTE-OFF* to be sent.

Bar-and-Note Storage

So far, I have discussed playing notes but not rests. Of course, rests are simply the spaces between notes, but the storage formats outlined do not provide a way of tracking down these spaces for correlation with written music. Rests can be stored in the same way as notes, but note numbers and velocities are not needed. This seems to be a reversion to the original relative timing method when you consider that rests are similar to the old embedded relative timing markers. Now the *NOTE-ON* time stamps become redundant—where one note or rest stops, the next will start. Without some frame of reference, though, it is difficult to find a designated spot in the data stream. I have borrowed another device from written music: bar lines. There is no MIDI equivalent for a bar line, so I use 0bah. The bar marker is followed by 1- or 2-byte bar numbers to allow absolute locations to be found. This combines some of the better features from all the methods outlined earlier. I use 0a0h as the token for a rest (see Example 5, below).

Some storage formats are better suited to certain approaches to editing or to certain looping constructs or display formats. Choose a format that suits your application, but remember to take as general an approach as possible. You will undoubtedly want to expand later to incorporate new ideas.

Quantization

I first mentioned quantization in the context of scaling down the system clock. If a section of music was recorded using a clock with 96 pulses per quarter note, the system clock could simply be slowed to 24 pulses per quarter note. Each timer interrupt would then increment the system timer by 4 instead of 1. This method works, but it has one main drawback. If there is no frame of reference (an unquantized track, for instance), it may not be noticed but all notes will effectively be shifted late by the quantize interval. This is like truncating a number when the real intention is to round it.

To quantize events so that the notes fall on the beat rather than after the beat, the timing target must be anticipated by half the amount of the quantization period. This causes

```
---94h 09h 03h 46h 32h -------------- 84h 08h 04h 46h 16h -------
   |output a NOTE-ON on ch.4             |output a NOTE-OFF on ch.4
   |     on the ninth clock tick         |     on the eighth clock tick
   |         of the third beat           |         of the fourth beat
   |             for note no. 46h        |             for note no. 46h
   |                 velocity = 32h      |                 OFF velocity = 16h
```

Example 3: *End-point-absolute storage*

```
--94h 09h 03h 46h 32h -------------- 04h 01h ---------------------
  output a NOTE-ON on ch.4              These are duration bytes that
      on the ninth clock tick           travel with the note record and are
          of the third beat             incremented by each timer tick
              for note no. 46h          until NOTE-OFF is received.
                  velocity = 32h
```

Example 4: *Single-point-absolute storage*

```
-0bah 32h -----0a0h 02h 14h ----------- 94h 46h 32h ---------------01h 02h---
| At bar #32h | rest                    | output a NOTE-ON on ch.4  duration
|             |      for two beats      |    for note no. 46h        is 1 beat
|             |          and 14h ticks  |         velocity = 32h     and 2 ticks
```

Example 5: *Bar-and-note storage*

(continued from page 37)

events to be processed in the center of a quantize window. To accomplish this, the system timer contents are copied to a look-ahead timer and incremented so that it leads the actual system time by half the quantize interval. Using look-ahead timers for compares will trigger events ahead of time. Remember just to run the clock routine every Nth clock tick and to add N/2 to the current time to derive the look-ahead timer.

The same result can be accomplished by first running the clock routine N/2 times consecutively (this accounts for look-ahead). After this initial look-ahead, the clock cycle consists of waiting for N clock periods and then running the clock routine N times consecutively.

Quantization cleans up notes whose timing is slightly frayed at the edges, but some mistakes can actually be accentuated. If the mistake is severe enough to fall outside the intended quantize window, note timing will be rounded in the wrong direction, as in note 3 of Figure 2, left.

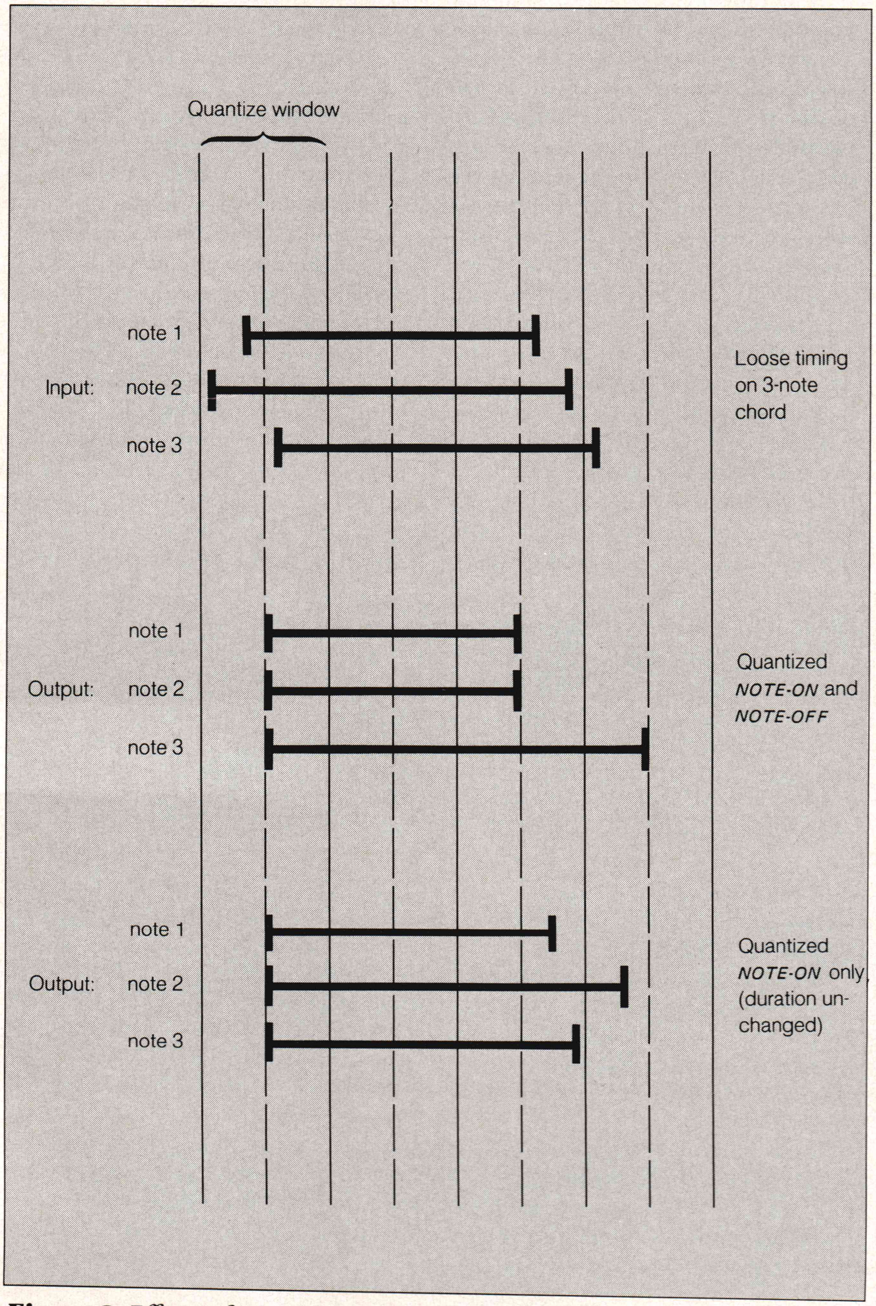


Figure 2: *Effects of quantization on note durations*

I mentioned that the trailing edges of notes are usually much less timing critical than the leading edges. One of my favorite techniques for avoiding a mechanical sound is quantizing the leading edges of notes but leaving the lengths intact. This can be accomplished by using a look-ahead on the clock that pulls the *NOTE-ON* event and note length from the data stream. The unprocessed note length is stored in a time-out list, where it is decremented on each tick of the high-resolution clock. The appropriate *NOTE-OFF* is sent when it reaches zero.

The timing of recorded music is easily altered or quantized, but random timing information from human input is difficult to add later (you could try it). Some synthesizer systems, such as Fairlight, use high-performance hardware to derive timing clocks as high as 384 clocks per quarter note, but be wary of micro-based software that claims this level of accuracy. By attempting performance beyond the capability of the machine, the software can actually sacrifice accuracy.

Pilot Track

Most musical compositions have verses, choruses, and other types of sections. Sections are usually recorded or written separately. When all the sections are completed, they are rearranged, repeated, and so on by the use of what I call a pilot track. The pilot track in this type of system operates outside the time frame of the composition. In fact, it may be the only timing stream that moves in a linear fashion. Macro commands, such as *play section 3, five times*, can pilot the interpreter through the appropriate series of pointer loads, plays, and reloads to accomplish the task. (See Figure 3, page 39.)

The start of each section now becomes the main point of reference for note timing. At the end of a section, the system timer is usually reset and the pilot track is consulted to find the next section to be played. Each section is treated as if it were a separate composition, so each can continue up to the maximum length allowed by the system timer.

Some pilot-track schemes use a Forth-type stack to hold reiterative

constructs and section or data references. I have even seen a sequencer that allows English statements such as *SECTION 3 = VERSE 1 + CHORUS.* In any case, the section is treated as a subroutine, with control returning to the pilot track when the section has completed.

Advanced Features

If you've made it this far, you may be interested in some of the extensions to MIDI protocol that allow more rapid transmission of event triggers. Running status says that the receiver keeps the most recently received command byte. If additional data bytes are received without a leading command byte, the old command byte is used.

To transmit *NOTE-ON*s on channel 3 for note numbers 22h, 33h, and 44h with velocities of 55h, 66h, and 77h, the bytes shown in Example 6, page 41, could be sent. Duplicating the

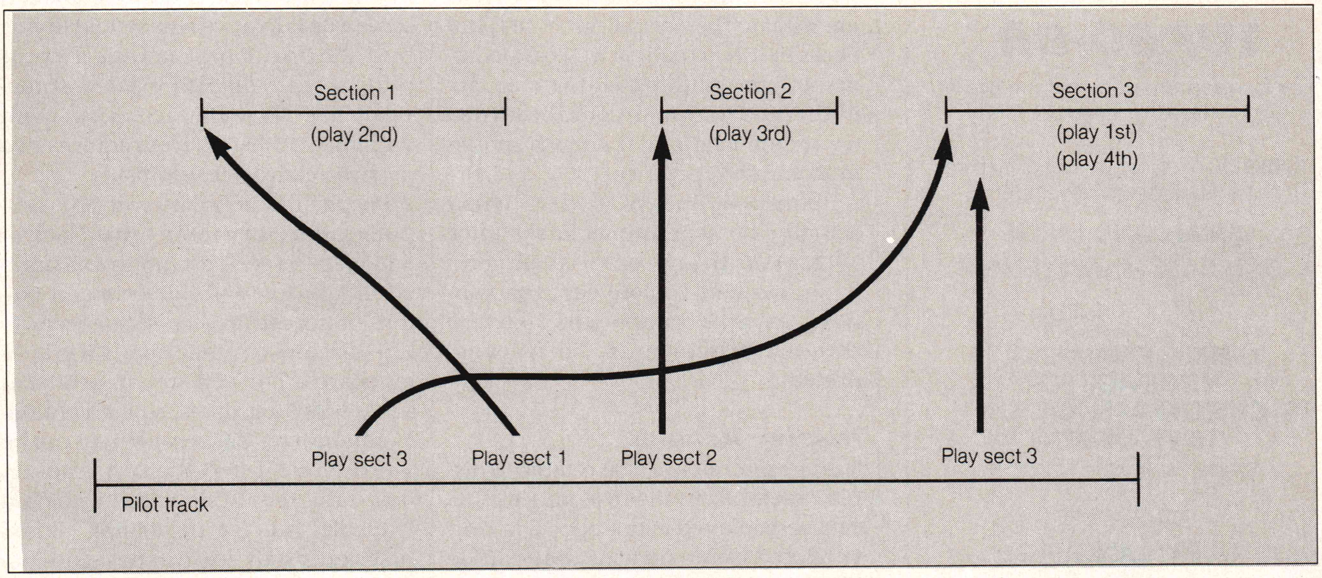


Figure 3: *Relating the pilot track to previously stored sections of a composition*

(continued from page 39)

same series with an 83h as the first byte turns the notes off again.

To make this feature more useful, the special condition *NOTE-ON with velocity = 0* is reserved to signal a *NOTE-OFF* operation. Its function is identical to the normal *NOTE-OFF*, but because it is actually a *NOTE-ON* command, the running status rule applies. The same notes could be turned on and off again by the bytes shown in Example 7, page 41.

Remember that *NOTE-OFF*s won't actually be sent immediately after *NOTE-ON*s. If any other command bytes are sent in between, the running status is interrupted and the command byte must be retransmitted.

Display Methods

Storage methods can be related to display methods in that the note events can be displayed at one spot (as a conventional music note) or they can be displayed as a (usually horizontal) band on the screen with the positions of the start and end points representing the start and end times of the note. The latter method, sometimes known as piano-scroll notation (because of the similarity to a player piano scroll), is analogous to the endpoint storage method I have outlined. So, do software designers who use single-point storage use conventional notation and designers who use endpoint storage use piano-scroll notation? Of course not. Strangely enough, some of the more popular software packages use exactly the opposite techniques from those you'd expect.

Both display methods have advantages: piano-scroll notation can be visible to musicians who are not accustomed to reading music, whereas conventional notation maintains high information density. Also, conventional notation always requires graphics capability. Piano-scroll can usually be done with text-mode graphics.

Displaying notes on staff lines requires at least 4 or 5 vertical pixels per line on the staff, for a total of 17–21 pixels for a complete staff line (4 lines times 4–5 pixels, plus an extra line). I have seen sheet music with notes printed as far as six spaces above or below the staff, so it is wise to allow a lot of blank space on each side of a staff line. Allowing 60 vertical pixels total per staff should yield 5 or 6 staff lines on a high-resolution screen.

Horizontal formats are usually best handled and allocated by the byte. A screen 640 pixels across would divide into 80 horizontal compartments with screen objects treated somewhat like ASCII characters but with variable widths (a G clef requires two or three character widths).

Despite the popularity of IBM computers, the CGA's unfortunate lack of adequate screen resolution has limited its use for staff-line-oriented editors or forced earlier sequencers to use Hercules or other non-IBM graphics boards. The CGA screen produces square-looking notes, so conventional notation on this screen may not be worthwhile. I find the EGA graphics board extremely slow, but color is a valuable tool for displaying music. Notes on a staff line can be color-coded to designate channel numbers (to allow more than one channel per staff line). Monochrome editors sometimes allow selection between two channels per staff by directing note stems up or down.

Display formats will depend largely on the display devices you have on hand or wish to support. Although EGA boards have finally made the IBM PC competitive with other computers when displaying color, you may only want to enter musical notes and then print them out on paper. Some music transcribers are using Jim Miller's software without ever buying MIDI hardware for their computers. When doing black-and-white printouts, obviously a Hercules mono graphics card will suffice. The Hercules board has 720 horizontal pixels, which ends up looking a lot longer than the EGA's 640 pixels (sometimes a full bar of music).

Note-event editors take many different forms, but list-oriented editors are the easiest to design, followed by piano-scroll editors. List editors can simply convert a list of note numbers into *NOTE-ON* and *NOTE-OFF* events. This may be adequate if the main function of the software is to record live music being played on a synthesizer.

Most piano-scroll editors use strict binding between screen position and note events. The screen is sometimes partitioned into an even bar of music, for example. The x-axis position of the cursor then relates directly to the music's time domain.

Staff-line editors usually require a complex series of pointers to correlate records in the data stream with locations on the screen. Using a cursor to locate and change a point within the music data stream can be a difficult task because the screen spacing may be nonlinear when related to the time domain (a bar consisting of a single whole note will be shorter than a bar that holds a series of 16th notes). Many staff-line editors do require vertical alignment of synchronized events, such as left-and right-hand piano parts that are written on separate staffs. This allows the screen to be swept from left to right with a single x-axis pointer, but strange timing errors are introduced by the converter when vertical alignment is not maintained. If you are writing this type of editor/converter, I recommend keeping a separate x-axis pointer for each staff line. Staff-oriented editors are further complicated by the need for multiple data representations. The on-screen symbols usually must be transformed into a format that is quite different in order to play them as MIDI notes.

A piano-scroll editor is a good starting point for experienced software writers who have limited knowledge of traditional musical concepts. Only the most experienced writers should attempt to write a staff-line editor, as it requires a thorough knowledge of both music theory and programming.

Writing Your Own Patch Editor

One of my current projects is a patch editor for the Kawai K-3 synthesizer, so I can explain exactly how patch editing works. The K-3 generates sound by building waveforms from sine-wave harmonics. These waveforms can be manipulated from the K-3's front panel, but the addition of a computer screen offers an enormous advantage in visualizing sounds as waveforms are being altered. In my wave editor, color-coded bars move to indicate harmonic numbers and amplitudes. As the operator tailors the waveform by adjusting the on-screen representation, an internal table of values is also adjusted. Just as in a text editor, different versions of the data can be saved to disk as the waveforms are being edited. The K-3 holds only 1 internal user-defined wave, but I hold up to 100 waves within one file so they can be compared and interchanged.

When a patch or wave is ready to be sent to the K-3, a MIDI *SYSTEM EXCLUSIVE* command tells the synthesizer to expect new patch information. No acknowledge is necessary—the harmonic numbers and amplitudes are sent out immediately. Transmitted data can be of any length, but 8-bit data must be sent as two separate nybbles to ensure that the MSBs will be reset (0). Because of the potentially long data stream, Kawai requires a checksum for confirmation, but this is entirely up to the manufacturer. The data packet is closed by an *END OF EXCLUSIVE* byte, which lets the synthesizer get back to music processing. The bytes sent to the K-3 to set up a wave look like those shown in Example 8, below.

Data formats for send and for receive are identical, so two synthesizers can be hooked together for trading waveforms or waves can be sent to the computer, modified, and then sent back to the K-3.

Every manufacturer and synthesizer has a different format for system-specific commands. Consult synthesizer manuals for details.

Choosing a Synthesizer

If you will be testing your MIDI software by recording music in real time, you will need a keyboard or other source of MIDI note messages. If cost is a factor, look at Casio's CZ-101 (miniature keyboard) or CZ-1000 (larger version). In a slightly higher price bracket, the Kawai K-3 has a good performance-to-cost ratio. Yamaha has developed a wide range of FM instruments, and its DX-7 series is one of the best-selling series of synthesizers in the over-$1,000 range. KORG also makes several affordable machines.

```
93h-------------22h---55h---------33h---66h---------44h---77h------
|                |     |           |     |           |     |
NOTE-ON cmd     note#  vel        note#  vel        note#  vel
Ch 3            first  note       second note       third  note
```

Example 6: *Transmitting individual notes over* MIDI

```
93h-------22h--55h---33h--66h--44h--77h----22h---0----33h---0-----44h---0
|          |    |     |    |    |    |      |     |    |     |     |     |
NOTE-ON  note# vel  note# vel note# vel   note# vel  note# vel  note# vel
Ch 3    first_note second_note third_note first_note second_note third_note
```

Example 7: *A MIDI note stream*

```
-0f0h-40h-0h-20h-0h-1h-64h----0h-1h--0h-5h-...........--1h-9h--1h-0fh----0f7h-
  |     |  |   |   |  |  |      |      |       |         |      |        |
SYS-EX | CHAN | GRP | INTRN  HARM=1 AMP=5 (more-data)HARM=19h AMP=1fh  EOX
    KAWAI  FUNC#  K3  WAVE
```

Example 8: *Example data stream to initialize Kawai wave table*

MUSIC RECORDER
(continued from page 41)

If you are interested only in patch editors, or if you already own a MIDI keyboard, consider using a stand-alone sound generator. These are becoming more popular because one MIDI keyboard can control several slaved sound generators. Kawai has introduced a keyboardless version of its K-3 synthesizer, and Yamaha has just introduced a module (the FB-01) for less than $400. Low-cost, rack-mounted sampling units include the AKAI model S612 and the Ensoniq Mirage.

The eight-voiced FB-01 and the four-voiced Casio CZ-101 can both assign separate tone patches to each voice. In other words, a piano sound, a violin sound, and a horn sound can all be played at once. This is an advantage to software designers because the module can be made to respond like multiple synthesizers. It is difficult to assign or prioritize multiple voices with a single keyboard, so the Casio allows this *MONO-MODE* operation only when sounds are played and assigned by an external computer. The FB-01, of course, has no keyboard.

Synthesizer keyboards generally have nonweighted plastic keys. If you prefer a keyboard that feels more like an acoustic piano, you may want to invest in one of the higher-quality MIDI keyboard controllers, such as the Roland MKB-1000 or Yamaha KX-88. Because these are output-only keyboards (they have no sound-generation electronics), they must be used in conjunction with an external MIDI-controlled sound generator.

The sound of a synthesizer depends both on its electronics and on its programming, so it is impossible to categorize every type of instrument. There are some guidelines, however. Oscillators can be analog or digital. The difference is somewhat like comparing records and compact discs: digital oscillators are precise and they can produce a wide range of timbres, but some say that they lack the warmth of analog oscillators. Analog machines, such as the Roland, Moog, or Oberheim synthesizers, are known for producing rich string patches or resonant brass sounds. Digital machines, such as the Yamaha line, excel at more percussive sounds, like pianos or bells. Samplers can capture breathy, human or flutelike voices. Musicians often use different types of synthesizers to cover different ranges of tonalities, but the ranges overlap quite a bit. Listen to as many patches as possible before making your choice.

SMPTE (and Other Time Codes)

Computerized recording is becoming popular, but the accepted medium for interchange of completed music is still audio tape. Tape is necessary for recording voices, guitars, and acoustic instruments and for transporting sounds between studios that have different types of synthesizers or drum machines. Synchronizing tape-based and computer-based recording media can be difficult. Fortunately, a solution to many of the problems already exists in the form of time code.

Time codes of various types have been in use for many years. They are used to lock audio recorders to video machines for movie sound tracks and for time stamping video tape for TV news. One of the most popular forms of time code was devised by NASA as a simple, fixed time reference for its experiments. It used recordable audio-range pulses in a format known as biphase modulation. Biphase uses clock transitions, rather than states of polarity, to encode binary data. This means that the output will never be a nonrecordable DC voltage. The 80-bit serial data stream encodes time as hours, minutes, seconds, frames, and subframes. The last two increments are arbitrary values that vary, depending on usage, but even this vague specification was sufficient to merit acceptance in a wide range of applications, especially video film. It has come to be known as SMPTE code, after the Society for Motion Picture and Television Engineers.

SMPTE code is based on a fixed time-of-day clock, rather than a variable rate, so it is not the ideal code for resolving the fine nuances of musical timing. Hardware synchronizers, incorporating complex frequency multipliers and phase-locking schemes, must be used to correlate MIDI tempo timing and SMPTE absolute timing.

MUSIC RECORDER

(continued from page 42)

Some of the first sync boxes for SMPTE-MIDI conversion were from Roland (SBX-80) and from Garfield Electronics. It is difficult to calculate rates and match-up points for the two time codes, so both of these units take the more practical approach of building a map of alignments as the time codes are received. The map is then stored to tape or to disk via MIDI.

An important step in the development of SMPTE-to-MIDI standards are the map formats being proposed by SMPTE synchronizer manufacturers such as Adams-Smith. Adams-Smith's new Zeta 3 system allows commands from MIDI or RS-232 to control a tape machine or lets time codes from the tape machine be translated back to MIDI format. Two tape transports and a variety of sequencers and computers can be operated from a single Zeta 3 synchronizer. In actual operation, machines are synchronized by striping one of the tape tracks on each machine with SMPTE code. The tape controller reads these tracks and fine-tunes motor speeds on the transports. The Zeta 3 controller also outputs MIDI timing bytes to keep sequencers in step with the tape.

If something is conspicuously absent on the sequencers you see, it is probably difficult to design.

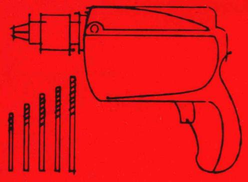
Another type of time code in common use is FSK, or frequency shift keying, which encodes 0s and 1s as two different frequencies. A major drawback to FSK is the lack of enough resolution to provide any form of embedded absolute time reference. FSK tapes must always be started from a known reference point because FSK is a relative timing reference.

Incorporation of SMPTE control or provision for some kind of sync-to-tape can be a big advantage when marketing software. It may only be necessary to stay compatible with support hardware marketed by other companies.

Summary

By providing a bridge between the music and computer industries, MIDI has sparked new interest in the design of innovative musical instruments. It has, in fact, created its own industry. Many competent software engineers are becoming interested in music because of this accessibility, and better products are being introduced every day.

Make sure you look at a few of the commercially available sequencers or patch editors before you start writing software. If you see something conspicuously absent on all the sequencers you encounter (such as a built-in universal patch editor), chances are that it is difficult to design. There are some imaginative designers working with MIDI, and some impossible things can be accomplished with a new approach or just a lot of work. Visit one of the larger music stores to find out what is currently on the market.



POWER TOOLS for SYSTEM BUILDERS™

1-800-543-6277

Ask for Operator 2053
California: 1-800-368-7600

TSF is owned and operated by programmers, so we understand your needs. We believe and practice integrity in all business dealings. We advertise real prices and offer the best possible terms to all customers. We provide prompt delivery of current product versions.

We carry only products which we have personally used or which come with strong recommendations from developers we respect. We have the technical capability and the interest to search for and find new products to meet your specific needs. We will gladly provide quotes for volume purchases or for products outside our normal product line. We consider it our job to help you get answers when publisher's technical support is unresponsive. At TSF service means more than a pushy sales pitch and a $2 lower price.

We accept checks, Visa, Mastercard, American Express and COD. We charge your card only when we ship and do **not** add a surcharge. We provide free UPS delivery on software orders over $100 ($3 delivery charge on orders under $100). Our COD fee is $4. We allow return privileges on most products. Give us a try.

The Software Family

649 Mission Street
San Francisco, CA 94105
(415) 957-0111
1-800-543-6277 (U.S.)
1-800-368-7600 (Calif.)
Ask for Operator 2053 on toll free calls

New! Supercharge Turbo Pascal

Mach2 from MicroHelp enhances Turbo Pascal with instant screen updates, windows, Ctrl-Break and DOS critical error trapping, flexible scrolling, Lotus-style menus and much more. Most functions are written in assembler for lightning fast operation and low memory overhead. Mach2 gives Turbo Pascal programs the i/o performance and hardware error handling capabilities of well-written assembly language programs. No royalties. All source included. (List $69) Only $55 from TSF. 30 day money back guarantee.

New! Create Demos AND Applications

Screen Machine from MicroHelp lets you paint screens to create demonstration prototypes for selling and refining your program concept. Any PC screen can be captured as a starting point or you can work from scratch. After finalize the concept, Screen Machine generates screen display code in Basic, Turbo Pascal, dBase II or III, or assembler (including support for Turbo Pascal and Basic Mach2 if desired). No royalties. All source included. (List $79) Only $63 from TSF. 30 day money back guarantee.

New! Visual Command Language

Opal from The Software Factory lets you overcome MSDOS batch limitations. Run any sequence of MSDOS programs using control logic based on the system date and time, operator entry (including comprehensive prompting and editing), the existence of files or the contents of files. A comprehensive but easy to use screen painter allows you to quickly generate menus providing cursor-key item selection with full control of display attributes. Complete with screen painter. Opal provides a cost effective alternative to custom written shells. Only $79. 30 day money back guarantee.

New! Professional Hardware Diagnostics

Almost every PC user has wasted time and money due to mis-diagnosis, unnecessary hardware service calls or technicians arriving without necessary parts. Windsor Technologies now provides comprehensive, menu driven diagnosis software with detailed problem reports in English. With the Windsor tools you can easily identify failed components and make sure that service technicians are called only when needed and that they know in advance what parts to bring. The service kits are used many national repair companies, work for IBM, clone and add-on components and inlcude terminators for loop-back testing of serial and parallel ports. The Windsor diagnostics let you provide the hardware expertise clients expect from you. Model T01 for PC/XT $195. Model T51 for AT $245. Model T91 for PC/XT/AT $395. 30 day money back guarantee.

Send Or Call For Comprehensive Free Catalog

TSF carries a complete line of software and hardware tools for programmers and system engineers. We offer "big name" products at competitive prices and a large selection of hard-to-find products that save you hours of research, evaluation and development work. Our catalog includes hype-less product descriptions that explain what our products do to improve your productity and your product's quality. Call or send the coupon for your free copy.

__ Please send me a free catalog

__ Please send the following products: ____________

Name: ____________
Address: ____________
City/State/Zip: ____________

Credit Card: ____________
Exp: ____________

Mail to: TSF, 649 Mission Street, San Francisco, CA 94105

CIRCLE 230 ON READER SERVICE CARD

MUSIC RECORDER
(continued from page 45)

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 3**.

Resources

User's Groups

IMA (Intl. MIDI Association)
12543 Hortense Ave.
Studio City, CA

IMUG (Intl. MIDI User's Group)
P.O. Box 593
Los Altos, CA 94022

SMPTE
(Society of Motion Picture and Television Engineers)
595 W. Hartsdale Ave.
White Plains, NY 10607

Periodicals

Electronic Musician
2608 Ninth St.
Berkeley, CA 94710
Subscription Dept.:
5615 Cermak Rd., Cicero, IL 60650

Keyboards Computers and Software
299 Main St.
Northport, NY 11768

Keyboard Magazine
P.O. Box 2110
Cupertino, CA 95015

Vendors

Adams-Smith
34 Tower St.
Hudson, MA 01749
(617) 562-3801

AKAI Professional Div.
P.O. Box 2344
Fort Worth, TX 76113

Casio Inc.
15 Gardner Rd.
Fairfield, NJ 07006
(201) 575-7400

Doctor T's Music Software
220 Boylston, Ste. 306
Chesnut Hill, MA 02172
(617) 224-6954

E-Mu Systems Inc.
1600 Green Hills Rd.
Scotts Valley, CA 95066
(408) 438-1921

Ensoniq Corp.
263 Great Valley Pkwy.
Malvern, PA 19355
(215) 647-3930

Fairlight Instruments
2945 Westwood Blvd.
Los Angeles, CA 90064
(213) 470-6280

Garfield Electronics
P.O. Box 1941
Burbank, CA 91507
(213) 434-6643

Hybrid Arts Inc.
11920 W. Olympic Blvd.
Los Angeles, CA 90064
(213) 826-3777

Jim Miller (Personal Composer)
P.O. Box 648
Honaunau, HI 96726
(808) 328-9518

Kawai America Corp.
2055 E. University Dr., C.S. 9045
Compton, CA 90224-9045
(213) 534-2350

KORG USA Inc.
89 Frost St.
Westbury, NY 11590
(516) 333-9100

Kurzweil Music Systems Inc.
411 Waverly Oaks Rd.
Waltham, MA 02154-8464
(617) 893-5900

Lexicon Inc.
60 Turner St.
Waltham, MA 02154
(617) 891-6790

Mark of the Unicorn
222 Third St.
Cambridge, MA 02142
(617) 576-2760

Mimetics Corp.
P.O. Box 60238, Station A
Palo Alto, CA 94306
(408) 741-0117

Moog Electronics
2500 Walden Ave.
Buffalo, NY 14225-4799
(716) 681-7242

New England Digital Corp.
49 N. Main
P.O. Box 546
White River Junction, VT 05001
(802) 295-5800

Optical Media International
P.O. Box 2107
Aptos, CA 94301
(408) 662-1772

Oberheim, A Division of ECC Development Corp.
11650 W. Olympic Blvd.
Los Angeles, CA 90064
(213) 479-4948

Opcode Systems
444 Ramona St.
Palo Alto, CA 94301
(415) 321-8977

Passport Designs Inc.
625 Miramontes St., Ste. 103
Half Moon Bay, CA 94019
(415) 726-0280

Roland Corp. US
7200 Dominion Circle
Los Angeles, CA 90040-36477
(213) 685-5141

Sequential Circuits Inc.
3051 N. First St.
San Jose, CA 95134
(408) 946-5240

Voyetra Technologies
426 Mt. Pleasant Ave.
Mamaroneck, NY 10543
(914) 698-3377

Yamaha International Corp.
6600 Orangethorpe Ave.
Buena Park, CA 90622
(714) 522-9011

Dimensional Data Types

by Do-While Jones

Back in the days when memory was expensive, computers were slow, and compilers weren't smart enough to optimize code, it was necessary to write concise, clever programs. The resulting programs were efficient, but they were cryptic, which made it difficult to modify the code without adding new bugs. Managers prized their few brilliant programmers who could write and maintain efficient code.

But then memory became cheaper, computers ran faster, compilers became better, and genius programmers started changing jobs whenever someone offered them more money. Managers discovered they were spending more money on software than they were on hardware.

The old values have changed. Compact code is no longer necessary or desirable. Instead, managers are looking for code that can be easily understood. This makes the code cheaper to validate and maintain and makes it possible for a team of programmers to work together efficiently and get their software product to the market (or battlefield) first. People who write programs that are so complex that nobody else can understand them are no longer an asset to an organization. The big money is now starting to go to people who can write clear code.

One way to make programs easier to understand is to use dimensional units (centimeters, grams, seconds, and so on) as data types. The bibliography lists some articles that show that there has been some interest lately in extending the syntax of high-level languages to include dimensional units.

Do-While Jones, 324 Traci Lane, Ridgecrest, CA 93555. Do-While is currently teaching Ada programming. He is a columnist for the Journal of Pascal, Ada, & Modula-2.

The big money now goes to those who write clear code.

You don't have to wait for someone to invent a new high-level language with built-in dimensional data types because one already exists. The Ada programming language has features that make it possible to invoke dimensional data types simply by referencing a library unit. But before I show you how, let me take a little time out to show you why people feel a need for dimensional units.

The Need for Dimensional Data Types

Example 1, page 51, shows a simple Ada program that could be used in a microprocessor-controlled radar speed gun. The program as it stands is perfectly legal, and presumably it would work if I added some instructions that really measured frequencies and showed the results on a display. This program is an example of bad programming practice, though, because it is difficult to validate and maintain.

If someone handed you Example 1 and asked you to determine if it were correct, could you? If someone asked you to change it so it gave answers in feet per second, how difficult would it be for you to make the change? What makes this program difficult to validate and maintain is the cryptic number *335,300,000*. Where did it come from? What does it mean? Is it correct?

Example 1 is too short to do justice to the problem, though, as it contains only one equation. You can concentrate your attention on that one line, and if you can't figure it out, you can rewrite the whole program and you haven't lost much. In the real world, a tactical embedded computer program has dozens, or hundreds, or maybe thousands of equations. This complexity makes it much more difficult to figure out, and rewriting the program from scratch is out of the question. Although Example 1 doesn't show the magnitude of the problem, I hope it gives you an idea of the kind of difficulties that you can encounter when ambiguous data types such as *float* are used.

How Dimensional Data Types Help

If you write the program using dimensional units as data types, the ambiguity problem goes away. Not only that, the compiler can catch obvious mistakes at compile time.

Even though Example 1 uses descriptive variable names and there is no question in your mind what *SPEED* represents, you don't know if it is calculated in feet per second, miles per hour, or kilometers per hour. On the other hand, if the objects in the program were declared using dimensional units, you would know everything you needed to

know about them—for example:

```
TRANSMIT_FREQUENCY,
        DOPPLER_FREQUENCY : Hertz;
SPEED : Miles_per_hour;
```

The Ada programming language allows you to derive new data types from existing ones. Therefore you could say:

```
type Hertz is new float;
type Miles_per_hour is new integer;
```

Then, assuming you had also declared *RECEIVED_FREQUENCY* to be of type *Hertz*, Ada would let you write this statement:

```
DOPPLER_FREQUENCY :=
            RECEIVED_FREQUENCY -
            TRANSMIT_FREQUENCY;
```

But if you had declared *RECEIVED_FREQUENCY* and *TRANSMIT_FREQUENCY* to be of type *Megahertz* and *DOPPLER_FREQUENCY* to be of type *Kilohertz*, then Ada would have rejected the preceding statement. The error message would have shown a dimensional-unit error at compile time.

Simply deriving new data types from existing numeric types is a step in the right direction, but it doesn't get you as far as you want to go. If I merely defined *Hertz* to be a new kind of floating-point number, then Ada would think it could multiply two objects of type *Hertz* together and get *Hertz* (instead of *Hertz* squared). Similarly, if I divided *Hertz* by *Hertz*, I should get a dimensionless floating-point number—not a result in *Hertz*.

The preceding paragraph shows some of the differences between dimensional quantities and simple scalar (dimensionless) numbers. Ada lets me define additional properties of dimensional data types that go beyond the scalar operations, but it won't let me undefine the operations that are valid for scalar objects but illegal for dimensional quantities. There is no way I can tell Ada that *Hertz* times *Hertz* doesn't give me an answer in *Hertz*.

Therefore, it isn't a good idea to derive dimensional data types from numeric data types. Fortunately, there is another way.

A Better Solution

Listing One, page 58, shows an Ada package that creates dimensional data types. Packages are standard Ada constructs that generally come in two parts. The package specification defines the available services, and the package body tells Ada how to implement those services.

The *DIMENSIONAL_UNITS* package specification defines two private data types called *Integer_unit* and *Float_unit*. As these are private types, they have only three legal operations:

- *Assignment*— You can assign a value to an object of this type.
- *Equality*— You can check to see if two objects of this type are identical.
- *Inequality*— You can check to see if two of these objects differ in any way.

All three of these operations are valid and useful for dimensional quantities, but clearly more operations need to be provided.

You can add two objects measured in feet and you will get an answer in feet, so addition must be defined. You can multiply an object in feet by a dimensionless number, and the result will be in feet. If you divide feet by feet you get a dimensionless number. You can check to see if one variable measured in feet is shorter than a second variable measured in feet. You will find all these operations (and more) in the *DIMENSIONAL_UNITS* package specification.

The *DIMENSIONAL_UNITS* package body is simple but lengthy. Fortunately, you have to compile it only once. Then it becomes part of your bag-of-tricks library, and you can access it with a one-line context clause. (Modern software professionals are trying to promote this kind of universal, reusable code.)

After I wrote this package, I started building a package called *WEIGHTS_AND_MEASURES* that would derive all possible dimensional units. It started out like this:

```
with DIMENSIONAL_UNITS; use
                    DIMENSIONAL_UNITS;
package WEIGHTS_AND_MEASURES is

    type Inches is new Integer_unit;
    type Feet is new Integer_unit;
    type Yards is new Integer_unit;
    type Miles is new Integer_unit;
```

```
--                    BADGUN.ADA


procedure Bad_Example is

  TRANSMIT_FREQUENCY, DOPPLER_FREQUENCY, SPEED : float;

  function Xmit_Frequency_Measurement return float is
  begin
    return 1.0; -- Machine specific code to measure frequency
                -- goes here
  end Xmit_Frequency_Measurement;

  function Doppler_Frequency_Measurement  return float is
  begin
    return 1.0; -- Machine specific code to measure frequency
                -- goes here
  end Doppler_Frequency_Measurement;

  procedure put(N : integer) is
  begin
    null; -- Machine specific code to display an integer goes
          -- here
  end put;

begin
  TRANSMIT_FREQUENCY := Xmit_Frequency_Measurement;
  DOPPLER_FREQUENCY := Doppler_Frequency_Measurement;
  SPEED := 335.30e6 * DOPPLER_FREQUENCY / TRANSMIT_FREQUENCY;
  put(integer(SPEED));
end Bad_Example;
```

Example 1: *An Ada program that demonstrates bad programming practice*

DIMENSIONAL DATA TYPES
(continued from page 51)

```
type Centimeters is new
                    Integer_unit;
.
.
.
end WEIGHTS_AND_MEASURES;
```

My intention was to start every program that needed dimensional quantities with a context clause invoking *WEIGHTS_AND_MEASURES*. A typical program would have looked like this:

```
  with WEIGHTS_AND_MEASURES; use
              WEIGHTS_AND_MEASURES;
procedure Main_Program is
  X, Y, Z : Feet;
begin
(do something with X, Y, and Z)
end Main_Program;
```

The problem was that there are too many dimensional units to list them all and define the relationship to every other related quantity. Just look through any physics reference book, and you will find pages of conversions from one kind of unit to another. Even after discarding units I knew I would never use (such as cubic furlongs), there were still far too many to make a universal *WEIGHTS_AND_MEASURES* package practical. So, I now derive just the units I need for each program.

An Example

Listing Two, page 61, shows the speed gun program rewritten using better style. It consists of five individual compilation units: *SPEED_GUN_UNITS* (specification), *HARDWARE_CIRCUITS* (specification), *Speed_Gun* (main program body), *SPEED_GUN_UNITS* (body), and *HARDWARE_CIRCUITS* (body).

The first compilation unit (*SPEED_GUN_UNITS*) is the small, customized version of *WEIGHTS_AND_MEASURES* for this application. It defines only three dimensional data types: *Miles_per_hour*, *Hertz*, and *Miles_per_second*. The main program computes a speed in *Miles_per_second* but the answer is desired in *Miles_per_hour*, so the *Type_Convert* function is provided in this package to make the conversion. I didn't want to clutter the main program with this type conversion, so I defined a special multiplication function that includes an automatic conversion from *Miles_per_second* to *Miles_per_hour*. A package specification of this type often contains special arithmetic operators that convert data types—for example, a division operator that divides *Feet* by *Seconds* and returns a value of type *Feet_per_second* is common.

The second compilation unit (*HARDWARE_CIRCUITS*) separates all the implementation-dependent code from the main program logic. If this were a real project (rather than an classroom exercise), I could write and debug my speed calculation while being blissfully ignorant of how some other guy was writing the code to measure frequency and display numbers. (Two of us would have trouble doing this with Example 1.)

Programmers should no longer waste time combining constants—the compiler should do it.

The third compilation unit is the main program, and it looks a lot like Example 1, only easier to read. If I gave you the procedure *Speed_Gun* instead of Example 1, you could see at a glance how I am computing *SPEED*. To validate it, you would only need to satisfy yourself that it is a correct rearrangement of the usual Doppler frequency equation—fd = (2 × speed × fx) / (speed of light), where *fd* is the Doppler frequency and *fx* is the transmitted frequency.

If you compile the first three units in the order given, Ada will check for dimensional consistency automatically and tell you that there are no errors. Ada doesn't need units 4 and 5 until you ask it to link the modules to create an executable image.

The fourth compilation unit tells Ada how to implement the two functions defined in unit 1. Ada checks for consistency between the specification and the body to make sure they match. The functions are easily verified and could easily be tested separately from the *Speed_Gun* program.

Finally, the fifth unit shows how you can test software before the hardware is finished. I chose to simulate the inputs by prompting the user to enter some numbers from the terminal, but I could just as easily have made the unit read numbers from a disk file.

If someone were to really build the speed gun, all I would have to do would be to rewrite the *HARDWARE_CIRCUITS* package body (unit 5) and recompile it. Relinking *Speed_Gun* would then replace the terminal I/O with the new speed gun circuit interface. I would not have to change (or even recompile) the first four units. Note that the *HARDWARE_CIRCUITS* package could also be tested without *Speed_Gun*.

There are some test results at the end of Listing Two.

The Overhead Isn't as Bad as It Looks

Listing Two is longer than Example 1, and that might be a cause for some concern. To make the comparison fair, you have to ignore compilation unit 5 (because Example 1 leaves that part out), but even so it is clear there is some overhead (in terms of program lines) when using dimensional units. In this example the overhead is significant, but in practical programs it isn't as high. The overhead appears out of proportion here because the computational part of the example program is so trivial.

Consider this analogy. You might think it impractical to use a computer to keep track of your checkbook balance because the overhead would be too high. Every time you wrote a $10 check, you would have to remove the dustcovers, turn on the computer, wait for the CRT to warm up and the hard disk to come up to speed, boot the operating system, load the checkbook balancing program, enter the data, copy the result to your checkbook, turn off the computer, and put the dustcovers back on. It would be easier just to subtract the $10 on paper! But if you were running a bank, you wouldn't dream of keeping track of all of each customer's checking accounts with pencil and paper. The

DIMENSIONAL DATA TYPES
(continued from page 52)

same overhead would still exist (you would still have to take off the dust-covers and turn on the computer), but it would now be a small part of the whole process.

Practical embedded computer programs are so much more complicated than the speed gun example that the few extra lines needed to invoke a *WEIGHTS_AND_MEASURES* package are negligible.

Listing Two Can Be as Efficient as Example 1

Example 1 simply multiplies a ratio by a predetermined conversion constant, but Listing Two appears to compute that conversion constant at run time by dividing the speed of light by 2 and then multiplying by 60 twice. If it really did that, Listing Two would run much slower than Example 1. An old FORTRAN-IV compiler might have optimized Listing Two by storing the speed of light and the numbers *2* and *60* in registers instead of memory to make the program faster. Modern compilers are smarter than that. A good Ada compiler can recognize that all those constants can be combined at compile time and should generate exactly the same code for Listing Two as it would for Example 1 (if the terminal I/O simulation code was added to Example 1). Programmers should no longer waste time combining constants because the compiler should do it anyway. Combining constants just obscures the source code.

Conclusion

Listing Two probably seems radical to those who first learned to program a computer in the 60s (as I did). But we have to realize that the old values have changed. Now the most important feature of a program is that it be easily understood so it can be easily debugged, validated, and maintained. The use of dimensional units as data types is an important technique to adopt because it helps preserve the sense of the program.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600, extension 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

Bibliography

Jones, Do-While. "Readers' Forum." *Journal of Pascal, Ada, & Modula-2*, vol. 4, no. 1 (January/February 1985).
Manner, R. *SIGPLAN Notices* (March 1986).
Swaine, Michael. "Swaine's Flames." *DDJ* (August 1986).

DDJ

(Listings begin on page 58.)

Vote for your favorite feature/article.
Circle Reader Service **No. 4.**

SAS Institute Inc. Announces

Lattice C Compilers for Your IBM Mainframe

Two years ago...

SAS Institute launched an effort to develop a subset of the SAS® Software System for the IBM Personal Computer. After careful study, we agreed that C was the programming language of choice. And that the Lattice® C compiler offered the quality, speed, and efficiency we needed.

One year ago...

Development had progressed so well that we expanded our efforts to include the entire SAS System on a PC, written in C. And to insure that the language, syntax, and commands would be identical across all operating systems, we decided that all future versions of the SAS System—regardless of hardware—would be derived from the same source code written in C. That meant that we needed a C compiler for IBM 370 mainframes. And it had to be good, since all our software products would depend on it.

So we approached Lattice, Inc. and asked if we could implement a version of the Lattice C compiler for IBM mainframes. With Lattice, Inc.'s agreement, development began and progressed rapidly.

Today...

Our efforts are complete—we have a first-rate IBM 370 C compiler. And we are pleased to offer this development tool to you. Now you can write in a single language that is source code compatible with your IBM mainframe and your IBM PC. We have faithfully implemented not only the language, but also the supporting library and environment.

Features of the Lattice C compiler for the 370 include:

- **Generation of reentrant object code.** Reentrancy allows many users to share the same code. Reentrancy is not an easy feature to achieve on the 370, especially if you use non-constant external variables, but we did it.
- **Optimization of the generated code.** We know the 370 instruction set and the various 370 operating environments. We have over 100 staff years of assembler language systems experience on our development team.
- **Generated code executable in both 24-bit and 31-bit addressing modes.** You can run compiled programs above the 16 megabyte line in MVS/XA.
- **Generated code identical for OS and CMS operating systems.** You can move modules between MVS and CMS without even recompiling.
- **Complete libraries.** We have implemented all the library routines described by Kernighan and Ritchie (the informal C standard), and all the library routines supported by Lattice (except operating system dependent routines), plus extensions for dealing with 370 operating environments directly. Especially significant is our byte-addressable Unix®-style I/O access method.
- **Built-in functions.** Many of the traditional string handling functions are available as built-in functions, generating in-line machine code rather than function calls. Your call to move a string can result in just one MVC instruction rather than a function call and a loop.

In addition to mainframe software development, you can also use our new cross-compiler to develop PC software on your IBM mainframe. With our cross-compiler, you can compile Lattice C programs on your mainframe and generate object code ready to download to your PC.

With the cross-compiler, we also offer PLINK86™ and PLIB86™ by Phoenix Software Associates Ltd. The Phoenix link-editor and library management facility can bind several compiled programs on the mainframe and download immediately executable modules to your PC.

Tomorrow...

We believe that the C language offers the SAS System the path to true portability and maintainability. And we believe that other companies will make similar strategic decisions about C. Already, C is taught in most college computer science curriculums, and is replacing older languages in many. And almost every computer introduced to the market now has a C compiler.

C, the language of choice...

C supports structured programming with superior control features for conditionals, iteration, and case selection. C is good for data structures, with its elegant implementation of structures and pointers. C is conducive to portable coding. It is simple to adjust for the size differences of data elements on different machines.

Continuous support...

At SAS Institute, we support all our products. You license them annually; we support them continuously. You get updates at no additional charge. We have a continuing commitment to make our compiler better and better. We have the ultimate incentive—all our software products depend on it.

For more information...

Complete and mail the coupon today. Because we've got the development tool for your tomorrow.

SAS®

SAS Institute Inc.
SAS Circle, Box 8000
Cary, NC 27511-8000
Telephone (919) 467-8000 x 7000

I want to learn more about:

☐ the C compiler for MVS software developers
☐ the C compiler for CMS software developers
☐ the cross-compiler with PLINK86 and PLIB86

today...so I'll be ready for tomorrow.

Please complete or attach your business card.

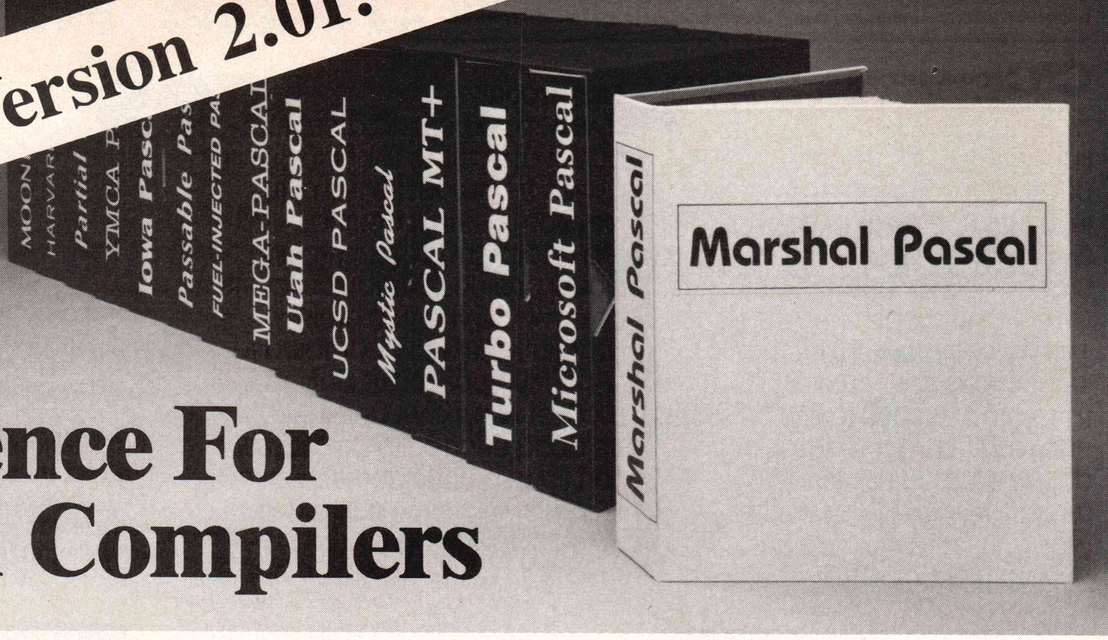
Name ____________________
Title ____________________
Company ____________________
Address ____________________
City ____________ State ______ ZIP ______
Telephone ____________________

Mail to: SAS Institute Inc., Attn: CC, SAS Circle, Box 8000, Cary, NC, USA. 27511-8000. Telephone (919) 467-8000, x 7000

DDJ 5/87

NOW–Version 2.01!

A New Reference For Pascal Compilers



	Ackerman		Sieve		I/O		Gauss-Seidel		Floating Point	
Marshal Pascal	**11.9**	**3.5K**	**4.8**	**2.4K**	**1.8**	**5.1K**	**4.9**	**11.5K**	**10.5**	**3.2K**
IBM Pascal	12.4	34.7K	11.7	27K	2.5	24.5K	7.6	34.6K	10.5	21K
Turbo Pascal	22.7	11.6K	14.2	11.5K	2.2	12.5K	4.7	13.5K	28	11.4K
Oregon Pascal-2	18.2	13.9K	7.2	11.7K	2.5	22K	11.8	37.6K	139.7	26.8K
Microsoft C 4.0	15.9	9.3K	5.8	6.5K	1.9	8.9K	6.0	23.6K	33	19.6K
	Sec.	Code Size	Sec.	Code Size	Min.	Code Size	Sec.	Code Size	Sec.	Code Size

Unparalleled Speed and Power

Marshal Pascal™ is the most highly code-optimized Pascal compiler for PCs. Period. In fact, Marshal Pascal produces code so fast and compact that even the most efficient C compilers fall behind in performance. Also, Marshal Pascal is an ISO implementation, thus offering portability to other computer environments. You may address as much memory as your operating system allows and a variety of memory models are supported. Among the useful extensions included are: separate compilation of modules (both Modula-2 and Pascal forms), structured constants and structured function values, variable-length string types and procedural parameters.

Turbo Pascal® Translator/Optimizer

Our translator/optimizer shrinks Turbo Pascal software enormously. You can watch your present Turbo programs run in a fraction of their former time and code space!

8087-80287 Support

Marshal Pascal supports the Intel 8087-80287® math processors *inline*. If you don't have the math chip, then '87-287 code simulation is a provided option.

Microsoft Linkability

Marshal Pascal's true relocatable linker allows you access to the Microsoft family of languages and assemblers. A flexible object code librarian and powerful overlay capabilities are also included.

Powerful Compile Options

Marshal Pascal gives you a number of compile options, including an optimization by-pass for speedier compiles, I/O "fine-tuning", constant folds and a syntax evaluator just to name a few. A wealth of compile-time checks permits you to find the more subtle logic errors, reducing debugging time enormously.

Efficient Large Heap Model for AI Applications

Marshal Pascal gives users efficient dynamic allocation for symbolic processing in AI applications. Marshal Pascal also allows functions to return arbitrary structured types. This can be used to combine the efficiency of Pascal with the powerful functional programming style enjoyed by such languages as LISP.

The Price? $189, *includes everything.*

Supports PC-DOS®, MS-DOS®, CP/M-86®, Concurrent DOS®, and *soon* Xenix 286/386®!

To order your copy of **Marshal Pascal,** call: **(800) 826-2222**
In California: **(415) 947-1000**

Marshal Language Systems

1136 Saranap Ave., Ste. P, POB 2010, Walnut Creek, CA 94595

CIRCLE 317 ON READER SERVICE CARD

Listing One *(Text begins on page 50.)*

```
Listing One

--                         DUNITS.ADA
--                         VERSION 1

--                         1 JANUARY 1986
--                         DO-WHILE JONES
--                         324 TRACI LANE
--                         RIDGECREST, CA 93555
--                         (619) 375-4607

package DIMENSIONAL_UNITS is

-- This package provides useful parent types for derived
-- dimensional units. That is, it makes it possible to
-- do this:

--  type Feet is new Integer_Unit;
--  type Radians is new Float_Unit;

--  PI             : constant Radians := Type_Convert(3.14159);
--  TARGET_RANGE   : Feet;
--  ANGLE          : Radians;
--  REVOLUTIONS    : integer;

-- These derived data types will inherit all the operations
-- in the package below. These are all the operations which
-- make sense for dimensional quantities.

-- The modulo operation for Float_Units is provided to make
-- it easy to normalize angular measurements.

--  ANGLE := ANGLE mod (2.0 * PI);

-- The division operator for Float_Units which return INTEGERS
-- truncates toward zero (rather than rounding) to make it
                                                    consistant
-- with integer division, and it lets you do this:

--  REVOLUTIONS := ANGLE / (2.0 * PI);

type Integer_Unit is private;

function Type_Convert(X : integer) return Integer_Unit;
  -- Lets you assign values to dimensional objects.
  -- For example,

  -- TARGET_RANGE := Type_Convert(587);

function "+"(RIGHT : Integer_Unit)
  return Integer_Unit;
function "-"(RIGHT : Integer_Unit)
  return Integer_Unit;
function "abs"(RIGHT : Integer_Unit)
  return Integer_Unit;
function "+"(LEFT, RIGHT : Integer_Unit)
  return Integer_Unit;
function "-"(LEFT, RIGHT : Integer_Unit)
  return Integer_Unit;
function "*"(LEFT : integer; RIGHT : Integer_Unit)
  return Integer_Unit;
function "*"(LEFT : Integer_Unit; RIGHT : integer)
  return Integer_Unit;
function "/"(LEFT : Integer_Unit; RIGHT : integer)
  return Integer_Unit;
function "/"(LEFT, RIGHT : Integer_Unit)
  return integer;
function "/"(LEFT, RIGHT : Integer_Unit)
  return float;
function "rem"(LEFT, RIGHT : Integer_Unit)
  return Integer_Unit;
function "mod"(LEFT, RIGHT : Integer_Unit)
  return Integer_Unit;
function Dimensionless(LEFT : Integer_Unit)
  return integer;
function Dimensionless(LEFT : Integer_Unit)
  return float;

-- "=" and "/=" are already defined for private types
function "<"(LEFT, RIGHT : Integer_Unit)
  return boolean;
function "<="(LEFT, RIGHT : Integer_Unit)
  return boolean;
function ">"(LEFT, RIGHT : Integer_Unit)
  return boolean;
function ">="(LEFT, RIGHT : Integer_Unit)
  return boolean;

type Float_Unit is private;

function Type_Convert(X : float) return Float_Unit;
  -- Lets you assign values to dimensional objects.
  -- For example,

  -- ANGLE := Type_Convert(3.14159);

function "+"(RIGHT : Float_Unit)
  return Float_Unit;
function "-"(RIGHT : Float_Unit)
  return Float_Unit;
function "abs"(RIGHT : Float_Unit)
  return Float_Unit;
function "+"(LEFT, RIGHT : Float_Unit)
  return Float_Unit;
function "-"(LEFT, RIGHT : Float_Unit)
  return Float_Unit;
```

(continued on page 61)

Introducing

Quaid Analyzer

the tool that created CopyWrite

Now you can debug your own programs with a professional quality debugger - the one that unraveled every form of copy-protection used on the PC.

With the Quaid Analyzer, you can:

- ☐ See occurrences of any interrupt, with its meaning shown on the screen.
- ☐ View memory as text or instructions, scrolling as easily as you do with an editor.
- ☐ Run until a memory location or I/O port is changed.
- ☐ Protect your hard disk from accidental destruction.
- ☐ Analyze software without the source, even when it uses countermeasures to thwart tracing.
- ☐ See all stages of the boot load.

We kept the Quaid Analyzer off the market to avoid helping publishers with copy-protection. Now that copy-protection is gone, we can sell it to you.

The Quaid Analyzer is a software tool occupying 100K bytes. It runs on any IBM PC and most MS-DOS systems without hardware modification.

To order

Call **(416) 961-8243**

Quaid Analyzer $99 U.S.

All orders shipped at our expense within a day. All major credit cards accepted.

or return coupon to:
45 Charles St. East
Third Floor, Dept. 604
Toronto, Ontario. M4Y 1S2

Payment method MC-Visa-Amex-Diners-Check

Card No. ____________________

Expiry Date ____________________

Name ____________________

Address ____________________

City/State ____________________

Phone No. ____________________

Signature ____________________

Quaid Software Limited

Ask about Disk Explorer the program that takes over where Quaid Analyzer leaves off.

8031

FORTH DEVELOPMENT ENVIRONMENT

Take advantage of Bryte's tools to make your job easier:

- Bryte's development environment uses BRYTE-FORTH on the actual production hardware during product development. No emulators, no changes, no surprises.
- Optional PC-based cross-development tools use DOS files as microcontroller mass storage. These files can be used to generate compact EPROM images, detailed listings, and cross-references.

Why not start developing the Bryte way today?

BRYTE-FORTH 8031 EPROM (includes 130 page User's Manual)	100.00
Utility disk(s)	65.00*
Cross-compiler/Cross-assembler	235.00*
8031 unlimitted quan. license	1000.00*

* Includes complete source code

bryte computers, inc.

P.O. Box 46
Augusta, ME 04330-0046

207/547-3218

CIRCLE 387 ON READER SERVICE CARD

Parallel Programming for "C"

INTERWORK

A Concurrent Programming Toolkit

Interwork is a "C" program library which allows you to write your programs as a set of cooperating concurrent tasks. Very useful for simulation, real-time applications, and experimentation with parallel programming.

FEATURES

- Supports a very large number of tasks (typically more than 100) limited only by available memory. Low overhead per task results in very fast context switching.
- Provides a full set of inter-task communication (ITC) facilities, including shared memory, locks, semaphores, blocking queues, and UNIX*-style signals. Also has building blocks for constructing your own ITC facilities.
- Handles interrupts (DOS version) and integrates them into task scheduling. Supply your own interrupt handlers or block tasks on interrupts.
- Lets you trace task switches and inter-task communication.
- Comes with complete documentation including a user's manual and reference manual of commands.

Interwork is available for the following systems:

Hardware	*Operating System*	*Price*
IBM PC, XT, AT	PC-DOS 2.0 or later	$129
IBM PC AT	XENIX*	$159
DEC VAX*, SUN III	UNIX 4.2BSD	$249

PC-DOS version is compatible with DeSmet, Lattice, and Microsoft C compilers.
Please specify hardware and operating system when ordering. Shipping and handling included; COD orders add $2.50. Send check or money order to:

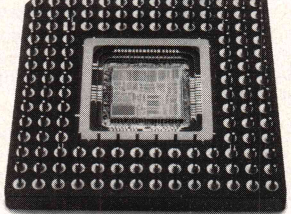
Block Island Technologies
Innovative Computer Software

13563 NW Cornell Road, Suite 230, Portland, Oregon 97229-5892
(503) 241-8971

*Trademarks: UNIX, AT&T Bell Laboratories, Inc.; XENIX, Microsoft, Inc.; VAX, Digital Equipment Corporation

CIRCLE 263 ON READER SERVICE CARD

A TRAINING COURSE FOR PEOPLE WHO LUST AFTER POWER.



The 80386

Intel is offering three new courses on the world's most powerful 32-bit microprocessor–the 80386. Plus a new intensive course on the 80286.

These in-depth learning sessions are designed for engineers and programmers who want to utilize the full power and potential of these lightning-fast chips.

Lectures are combined with hands-on workshops to provide real-life situations. Allowing you to apply new concepts and techniques immediately.

Courses include:

80386 System Software
80386 Programming using ASM386
High-end Microprocessor Hardware Design
The new 80286 Microprocessor Family Course

Complete training sessions and courses can be scheduled at your facility, or at our training centers.

In addition to training, Intel offers hardware/software support and consultants.

For more complete course information and schedules, call toll-free (800) 548-4725.

Or to register now, contact one of the Intel Training Centers listed below.

Intel Training Centers

Boston Area, Westford Corp. Ctr.
Three Carlisle Road, 1st Floor
Westford, MA 01886
(617) 692-1000

Chicago Area
300 N. Martingale Road, Suite 300
Schaumburg, IL 60194
(312) 310-5700

San Francisco Area
2700 San Tomas Expressway
Santa Clara, CA 95051
(408) 970-1700

Washington, D.C. Area
7833 Walker Drive, 5th Floor
Greenbelt, MD 20770
(301) 220-3380

intel®

Listing One *(Listing continued, text begins on page 50.)*

```
  function "*"(LEFT : integer; RIGHT : Float_Unit)
    return Float_Unit;
  function "*"(LEFT : Float_Unit; RIGHT : integer)

    return Float_Unit;
  function "*"(LEFT : float; RIGHT : Float_Unit)
    return Float_Unit;
  function "*"(LEFT : Float_Unit; RIGHT : float)
    return Float_Unit;
  function "/"(LEFT : Float_Unit; RIGHT : integer)
    return Float_Unit;
  function "/"(LEFT : Float_Unit; RIGHT : float)
    return Float_Unit;
  function "/"(LEFT, RIGHT : Float_Unit)
    return integer; -- trucates toward zero
  function "/"(LEFT, RIGHT : Float_Unit)
    return float;
  function "rem"(LEFT, RIGHT : Float_Unit)
    return Float_Unit;
  function "mod"(LEFT, RIGHT : Float_Unit)
    return Float_Unit;
  function Dimensionless(LEFT : Float_Unit)
    return integer;
  function Dimensionless(LEFT : Float_Unit)
    return float;

  -- "=" and "/=" are already defined for private types
  function "<"(LEFT, RIGHT : Float_Unit)
    return boolean;
  function "<="(LEFT, RIGHT : Float_Unit)
    return boolean;
  function ">"(LEFT, RIGHT : Float_Unit)
    return boolean;
  function ">="(LEFT, RIGHT : Float_Unit)
    return boolean;

-- The following don't have any application to dimensional
-- problems. I almost hid them in the package body, but I
-- thought that since I needed them to derive some of the
-- Float_Unit operations someone else might need them, too.

  function "/"(LEFT, RIGHT : float) return integer;
    -- divide and truncate toward zero

  function "rem"(LEFT, RIGHT : float) return float;

  function "mod"(LEFT, RIGHT : float) return float;

private

  type Integer_Unit is new integer;
  type Float_Unit is new float;

end DIMENSIONAL_UNITS;
```

End Listing One

Listing Two

```
--                    DUEX.ADA
-- This is an example of how the use of dimensional units as data
-- types improves program clarity.

------------------------ Compilation Unit 1 -------------------------

with DIMENSIONAL_UNITS; use DIMENSIONAL_UNITS;
package SPEED_GUN_UNITS is

  type Miles_per_hour is new Integer_Unit;
  type Hertz is new Float_Unit;
  type Miles_per_second is new Float_Unit;

  function Type_Convert(X : Miles_per_second)
   return Miles_per_hour;

  function "*"(LEFT : Miles_per_second; RIGHT : float)
   return Miles_per_hour;

end SPEED_GUN_UNITS;

------------------------ Compilation Unit 2 -------------------------

with SPEED_GUN_UNITS; use SPEED_GUN_UNITS;
package HARDWARE_CIRCUITS is
  function Xmit_Frequency_Measurement return Hertz;
  function Doppler_Frequency_Measurement return Hertz;
  procedure put(X : Miles_per_hour);
end HARDWARE_CIRCUITS;

------------------------ Compilation Unit 3 -------------------------

with HARDWARE_CIRCUITS; use HARDWARE_CIRCUITS;
with SPEED_GUN_UNITS; use SPEED_GUN_UNITS;
procedure Speed_Gun is
  TRANSMIT_FREQUENCY, DOPPLER_FREQUENCY : Hertz;
  SPEED : Miles_per_hour;
  C : constant Miles_per_second
   := Type_Convert(186_280.0); -- speed of light
begin
  TRANSMIT_FREQUENCY := Xmit_Frequency_Measurement;
  DOPPLER_FREQUENCY := Doppler_Frequency_Measurement;
  SPEED := (C / 2.0) * (DOPPLER_FREQUENCY / TRANSMIT_FREQUENCY);
  put(SPEED);
end Speed_Gun;

------------------------ Compilation Unit 4 -------------------------
```

(continued on next page)

Listing Two

(Listing continued, text begins on page 50.)

```
package body SPEED_GUN_UNITS is

  function Type_Convert(X : Miles_per_second)
   return Miles_per_hour is
    MPH : Miles_per_second;
  begin
    MPH := X * 60 * 60;
    return Type_Convert(Dimensionless(MPH));
  end Type_Convert;

  function "*"(LEFT : Miles_per_second; RIGHT : float)
   return Miles_per_hour is
  begin
    return Type_Convert(LEFT * RIGHT);
  end "*";

end SPEED_GUN_UNITS;

------------------------ Compilation Unit 5 ----------------

with TEXT_IO; use TEXT_IO;
package body HARDWARE_CIRCUITS is
  -- The statements below are standing in for code which would
  -- read the frequency directly from hardware circuits and
  -- would display speed on an LCD or LED display. Since I'm
  -- using a terminal as a substitute input device I used
  -- TEXT_IO to get and put data.

  package INT_IO is new INTEGER_IO(integer); use INT_IO;
  package F_IO is new FLOAT_IO(float); use F_IO;

  function Xmit_Frequency_Measurement return Hertz is
    F : float;
  begin
    put("What is the Transmit Frequency (in Hertz)? ");
    get(F);
    skip_line; -- TEXT_IO quirk
    return Type_Convert(F);
  end Xmit_Frequency_Measurement;

  function Doppler_Frequency_Measurement return Hertz is
    F : float;
  begin
    put("What is the Doppler Frequency (in Hertz)? ");
    get(F);
    skip_line; -- TEXT_IO quirk
    return Type_Convert(F);
  end Doppler_Frequency_Measurement;

  procedure put(X : Miles_per_hour) is
    I : integer;
  begin
    I := Dimensionless(X);
    put("The speed is "); put(I); put_line(" MPH.");
  end put;

end HARDWARE_CIRCUITS;

------------------------ Test Results ------------------------

$ run speed_gun
What is the Transmit Frequency (in Hertz)? 10.0e9
What is the Doppler Frequency (in Hertz)? 1600.0
The speed is          54 MPH.
$
$ run speed_gun
What is the Transmit Frequency (in Hertz)? 10.0e9
What is the Doppler Frequency (in Hertz)? 1000.0
The speed is          34 MPH.
$
$ run speed_gun
What is the Transmit Frequency (in Hertz)? 10.0e9
What is the Doppler Frequency (in Hertz)? 2000.0
The speed is          67 MPH.
$
```

End Listings

Listing Twenty-six

(Text in April)

```
/*----------------------------------------------------------------
 *
 *  NRPROCS.C
 *
 *  Copyright (c) 1986, Allen I. Holub. All rights reserved.
 *
 * Routines for processing individual commands, the following
 * commands cause a break unless ` is used as the command
 * character: .bp .br .ce .fi .in .nf .sp .ti
 */

#include <stdio.h>
#include <ctype.h>
#include "nr.h"

extern int      mgetc(), fgetc();
extern char     *skipspace(), *skipto(), *cpy();
extern char     *strsave ( char*  );
extern double   parse    ( char** );

static          Nestlev = 0;    /* .{/.} Nesting level  */

/*-------------------------------------------------------------*/

setnum( target, num, offset )
int     *target, num, offset;
{
        /* If offset is true, set target to its
         * previous value + num, otherwise set
         * it to num. Numbers are not allowed
         * to go negatative.
         */

        if( !offset )
                *target = (num < 0)  ?  0  : num  ;
        else
        {
                if( ( *target += num )  < 0 )
                        *target = 0;
        }
}

/*-------------------------------------------------------------
 * Routines to process individual commands: (note that the
 * comment command .\" is processed by expand() in nr.c.
 * The \" is the actual comment delimiter and a dot on a
 * line by itself is treated as a comment.
 */

sblock()
{
        /*  .{            --- NOT AN NROFF COMMAND ---
         *
         *  Starts a block for an .if, .ie, or .el.
         *  Bumps the process level up a notch. This
         *  routine can not be inhibited by "Inhibit."
         *  A \{\ or \{ is mapped to a .} for nroff
         *  compatability.                      (}})
         */

        ++Nestlev;
        process( Ifile, Ifilename, Ismacro, Macv );
        return 0;
}

/*-------------------------------------------------------------*/

eblock()
{
        /*   .}           --- NOT AN NROFF COMMAND ---
         *
         *  Terminate a .{ block
         *  Forces process() to terminate, bumping the
         *  nesting level back down a notch. This routine
         *  can't be inhibited by Inhibit. A \} is treated
         *  like a comment in terms of escape processing
         *  but it escape() will call eblock() in this
         *  case too. ({)
         *
         *  This command is not inhibitable
         */


        if( --Nestlev >= 0 )
            return 1;
        else
        {
            Nestlev = 0 ;
            err("Mis-matched .} (No corresponding .{ )\n");
            return 0;
        }
}

/*-------------------------------------------------------------*/

ad(lstr)
unsigned char   *lstr ;
{
        /* .ad [b n l r c]
         *
         *  Turn on adjusting. If *lstr is null then BOTH is
         *  used, otherwise the indicated adjustment mode is
         *  set.
         */

        Adjusting = 1;
```

```
        switch( *lstr )
        {
        case '\0':
        case BOTH:
        case ALT_BOTH:  Adjmode = BOTH ;
                        break;
        case LEFT:
        case RIGHT:
        case CENTER:    Adjmode = *lstr;
                        break;
        default:
            err("Bad mode: use (l)eft (r)ight (c)enter
                                    (b)oth=(n)ormal.\n");

        }
}

/*--------------------------------------------------------*/

cm( str )
char    *str;
{
        /*  .cm [on]     -- NOT AN NROFF COMMAND --
         *      enable nroff-style copy mode inside macro
         *      definitions. If no argument, nroff copy
         *      mode is disabled. In normal copy mode only
         *      \" and \<CR> are recognized. In nroff mode
         *      the following are recognized:
         *      \" \<cr> \n \* \$ \\ \. \t \a
         */

        Nr_cpmode = *str ;
}

/*--------------------------------------------------------*/

af(lstr, rstr)
char    *lstr, *rstr;
{
    /* .af R [1 001 i I a A e E]
     *
     *  Alter format of number register R to the
     *  indicated mode. Default is arabic.
     */

    register int        c;

    if( *lstr )
    {
        switch( c = *rstr )
        {
        case PADDED:

            while( isdigit(*rstr)  &&  c < '9' )
            {
                rstr++;
                c++;
            }

            if( isdigit( *rstr ) )
                err("Only 9 digits of zero fill allowed\n");

            break;

        case '\0':      c = ARABIC; break;
        case LC_ROMAN:
        case UC_ROMAN:
        case LC_ALPHA:
        case UC_ALPHA:
        case LC_ENG:
        case UC_ENG:
        case ARABIC:    break;

        default:
            err( "Illegal number register format <%c>\n", c );
        }

        putnreg(lstr, c , 0, -1, 0, 0 );
    }
}

/*--------------------------------------------------------*/

am(lstr, rstr)
char    *lstr, *rstr;
{
        /* .am xx yy
         *
         *      Append text to the macro named xx until
         *      either .. or .yy (in rstr) is found at the
         *      start of the line.
         */

        if( *lstr )
                mappend( lstr, rstr );
        else
                err("Missing macro name to .am\n");

}

/*--------------------------------------------------------*/

as(lstr, rstr)
char    *lstr, *rstr;
{
        /* .as lstr rstr
         *
         *      append rstr to end ofstring named in lstr
```

(continued on next page)

Listing Twenty-six *(Listing continued)*

```
         */

        if( *lstr )
                sappend( lstr, rstr );
        else
                err("Missing string name\n");
}

/*-----------------------------------------------------------*/

bd( lstr, rstr )
char    *lstr, *rstr;
{
        /* .bd on off    --- MODIFIED NROFF COMMAND ---
         *
         * Initialize bold face mode.
         * Send lstr to the printer to put it into bold mode,
         * send rstr to turn off boldface. Maximum length of
         * either string is 80 characters. Use \x to send
         * control characters.
         */

        static char     on[81], off[81] ;

        on[80] = off[80] = 0;

        strncpy( on,  lstr, 80 );
        strncpy( off, rstr, 80 );

        Bd_on  = on ;
        Bd_off = off;
}


/*-----------------------------------------------------------*/

bo( num, str, offset )
char    *str;
int     num, offset;
{
        /* .bo [ [+-]N ]          --- NOT AN NROFF COMMAND ---
         *
         *      Put the next N input lines into boldface.
         */

        setnum( &Num_bold, num, offset );
}

/*-----------------------------------------------------------*/

bp( num, str, offset, dobreak )
char *str;
{
        /*  .bp [ [+-] N ]
         *
         *      begin new page, having number N. If N is
         *      absent, use the current page number + 1;
         *      Note that N is applied to the new page,
         *      not the current one, so a footer on the
         *      current page will reflect the old number.
         */

        if( num )
                Nospace = 0 ;   /* Re-enable spacing    */

        if( dobreak )
                brk();

        prblank( (PGLEN-OLINE) + 1 );   /* Finish page  */

        if( num )
            PAGE = offset ? PAGE + num : num ;
}

/*-----------------------------------------------------------*/

br( str, dobreak )
char *str;
{
        /* .br  - stop filling and print current buffer.
         */

        if(dobreak)
                brk();
}

/*-----------------------------------------------------------*/

c2( lstr )
char    *lstr;
{
        /* .c2 [c]  - change nobreak character to c (*lstr)
         *      if c isn't defined, use `
         */

        Nobreak = *lstr ? *lstr : '`' ;
}


/*-----------------------------------------------------------*/

cc( lstr )
char    *lstr;
{
        /* .cc [c] -- make c the command character. If
         *              missing, use dot (.).
         */
```

```
        Cmd_chr = *lstr ? *lstr : '.' ;
}

/*------------------------------------------------------------*/

ce( num, str, offset, dobreak )
char    *str;
int     num, offset ;
{
        /* .ce [N]        -- Center the next N input lines
         *                without filling. Default N is 1.
         */

        if( dobreak )
                brk();

        Num_center = offset ? (Num_center + num) : num ;
}

/*------------------------------------------------------------*/

cf( str )
char    *str;
{
        /* .cf file   -- copy file directly to standard
         *                output. Useful for downloading
         *                fonts.
         */

        FILE            *fd;
        register int    c;

        if( !*str )
            err( "Missing filename in .cf\n" );
        else
        {
            if( !(fd = fopen(str,"rb")) )
                err(".cf %s, Can't open file\n", str);
            else
            {
                while( (c = getc(fd)) != EOF )
                    putc( c, stdout );

                fclose( fd );
            }
        }
}
/*------------------------------------------------------------*/

ch( num, str, offset )
char    *str;
int     num, offset;
{
        /* .ch xx [+-]N -- Change trap postion for macro xx
         *                 to N. Any existing trap at that
         *  position is destroyed (NROFF will shadow the
         *  earlier trap, not destroy it). If N is absent,
         *  the trap is removed.
         */

        if( *str )
                movetrap( str, num, offset );
        else
                err("Missing macro name in .ch command\n");

}
/*------------------------------------------------------------*/

cu( num, str, offset )
char    *str;
int     num, offset;
{
        /*  .cu [[+-]N]    -- Continuous underline next N
         *                input lines. All characters are
         *                underlined, even spaces.
         */

        setnum( &Cont_ul , num, offset );
}
/*------------------------------------------------------------*/

da( lstr )
char    *lstr;
{
        /* .da [xx]       -- Append to diversion xx. Stop
         *                appending when a .da or .di without
         *                an argument is encountered.
         */

        if( *lstr )
                dappend( lstr );
        else
                endiv();

}
/*------------------------------------------------------------*/

db( lstr )
char    *lstr;
{
        /* .db [1]        -- NOT AN NROFF COMMAND --
         *                Enable debugging mode (same as
         *                -v -c on the command line) if an
```

(continued on next page)

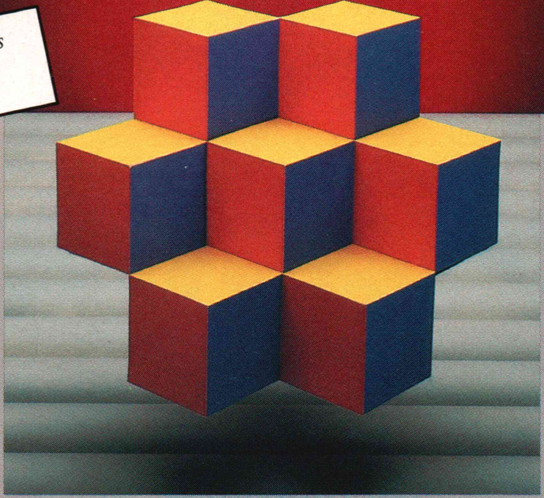
Listing Twenty-six *(Listing continued)*

```
         *               argument is present, else disable
         *               debugging mode.
         */

        Verbose = No_cntl = *lstr != '\0' ;
}

/*------------------------------------------------------------*/

de(lstr, rstr)
char    *lstr, *rstr;
{
        /*    .de xx yy  -- ENHANCED NROFF COMMAND --
         *
         *        Define a macro called xx. Definition stops
         *        when a .<rstr> is found at the beginning of
         *        If rstr is missing .. is used. If both
         *        arguments are missing, all currently defined
         *        macros are printed (like .pm in real nroff
         *        except contents of macro are printed too).
         *        If the macro already exists, it is deleted.
         *        Two copy modes are supported (see .cm).
         */

        if( *lstr )
                mcreate( lstr, rstr );
        else
                printm();
}

/*------------------------------------------------------------*/

df( lstr, rstr )
char *lstr, *rstr;
{
    /*  .df F <start> <end> <cwidths>
     *
     *                      -- NOT AN NROFF COMMAND --
     *
     * Define a font. F is a font name (one character),
     * <start> is a macro to invoke when font is invoked.
     * <end> is a macro to invoke when you switch out of
     * the font. <cwidths> is the name of a file that holds
     * the character-width tables (up to 255 char-sized
     * numbers delimited by whitespace or blank lines).
     * If no font name is specified then existing fonts
     * are printed to standard output.
     *
     * Lastfont (below) points at the most recently
     * added font. This routine assumes that main() will
     * call it to initialize the roman font before any
     * other fonts are defined. The behaviour is a little
     * strange though. findfont() always returns 0 when
     * the 'R' font is requested.  Consequently
     * findfont() won't return -1, for a nonexistant font,
     * when 'R' is defined. Be careful.
     */

    register FONT       *fp;
    register char       *p;
    static   FONT       *lastfont = &Fonts[0];
    int                 i, existing ;
    FILE                *stream;
    UCHAR               *malloc();

    if( !*lstr )
    {
        for( fp = Fonts-1; ++fp <= lastfont; )
        {
            printf("Font %c: start with <%s>, end with <%s>,
                                                    widths:",
                                 fp->name, fp->smac, fp->emac );
            p = fp->widths;
            for( i = 0; i < MAX_CHARS_IN_FONT; i++ )
            {
                if( i % 8 == 0 )
                    printf("\n");

                if( i < ' ' )
                    printf("^%c:%-3d ", i+'@', p[i]);
                else
                    printf("%2c:%-3d ", i    , p[i]);
            }

            printf("\n---------------------------------------\n");
        }
    }
    else
    {
        existing = findfont(*lstr);

        if( lastfont >= & Fonts[ NUMFONTS-1 ]   &&   existing < 0 )
            err("May not define more than %d fonts\n", NUMFONTS);
        else
        {
            if( existing < 0 )      /* Font doesn't exist       */
                fp = ++lastfont;
            else                    /* Redefining existing font */
            {
                fp = &Fonts[ existing ];
                if( fp->left )
                    free( fp->left );
            }

            fp->name        = *lstr ;
            fp->resolution  = Hs_amt;
```

(continued on page 72)

WHY LOGITECH MODULA-2 IS MORE POWERFUL THAN PASCAL OR C.

"A clear winner... The integrated editor is a joy to use."
BYTE Magazine, Jan. '87



APPRENTICE PACKAGE
$99

WIZARDS' PACKAGE
$199

- Separate Compilation w/inter-module typechecking
- Native Code Generation
- Large Memory Model Support
- Most Powerful Runtime Debugger
- Comprehensive Module Library
- Maintainability
- Translator from Turbo and ANSI Pascal

WIN A FREE TRIP TO Switzerland

HOMELAND OF MODULA-2

Return your Modula-2 Registration Card or a reasonable facsimile,* postmarked between March 1, 1987 and May 31, 1987 to be included in a once-only drawing!

Grand Prize: One week excursion for 2 in Zurich, Switzerland including a guided tour of ETH, the University where Modula-2 was created by Niklaus Wirth. European customers may substitute a trip to Silicon Valley, California.

Second and Third Prizes: LOGITECH C7 Mouse or LOGITECH Bus Mouse with Paint & Draw software—a $219 value, absolutely free!

*Write to Logitech, Inc. for a registration card facsimile.

NEW!

APPRENTICE PACKAGE $99

Everything you need to begin producing reliable maintainable Modula-2 code. Includes the Compiler with 8087 support, integrated Editor, Linker, and BCD Module. We're also including FREE our Turbo Pascal to Modula-2 Translator!

NEW!

WIZARDS' PACKAGE $199

This package contains our Plus Compiler—for professional programmers or for those who just want the best. The Plus Compiler with Integrated Editor requires 512K and takes advantage of the larger memory to increase compilation speed by 50%. Our Turbo Pascal to Modula-2 Translator is also included at no extra charge.

NEW!

MAGIC TOOLKIT $99

We've put our most powerful development tools into one amazing Toolkit for use with either the Apprentice or Wizards' packages. Highlighted by our Runtime Debugger, the finest debugging tool available anywhere, the Toolkit also includes our Post Mortem Debugger, Disassembler, Cross Reference utility and Version which keeps track of different versions of one program. Our MAKE Utility figures out module dependencies and automatically selects those affected by code changes to minimize recompilation and relinking. We also provide source code of our major library modules for you to customize—or just play with.

WINDOW PACKAGE $49

Now you can build true windowing into your Modula-2 code. Features virtual screens, color support, overlapping windows and a variety of borders.

ROM PACKAGE AND CROSS RUN TIME DEBUGGER $299

For those who want to produce rommable code. You can even debug code running in ROM from your PC.

Turbo Pascal is a registered trademark of Borland International.

Call for information about our VAX/VMS version, Site License, University Discounts, Dealer & Distributor pricing.

To place an order call toll-free:
800-231-7717
In California:
800-552-8885

YES! I want the spellbinding power of LOGITECH Modula-2!

☐ Apprentice Package	$99
☐ Wizards' Package	$199
☐ Magic Toolkit	$99
☐ Window Package	$49
☐ ROM Pkg/Cross RTD	$299

Add $6.50 for shipping and handling. Calif. residents add applicable sales tax. Prices valid in U.S. only.

Total Enclosed $______

☐ VISA ☐ MasterCard ☐ Check Enclosed

Card Number ____ Expiration Date ____
Signature ____
Name ____
Address ____
City ____ State ____
Zip ____ Phone ____

LOGITECH

LOGITECH, Inc.
805 Veterans Blvd. Redwood City, CA 94063
Tel: 415-365-9852

In Europe:
LOGITECH SA, Switzerland
Tel: 41-21-879656 • Telex 458 217 Tech Ch

In Italy:
Tel: 39-2-215-5622

CIRCLE 257 ON READER SERVICE CARD

"How to protect your software by letting people copy it."

By Dick Erett, President of Software Security

Inventor and entrepreneur, Dick Erett, explains his company's view on the protection of intellectual property.

"A crucial point that even sophisticated software development companies and the trade press seem to be missing or ignoring is this:

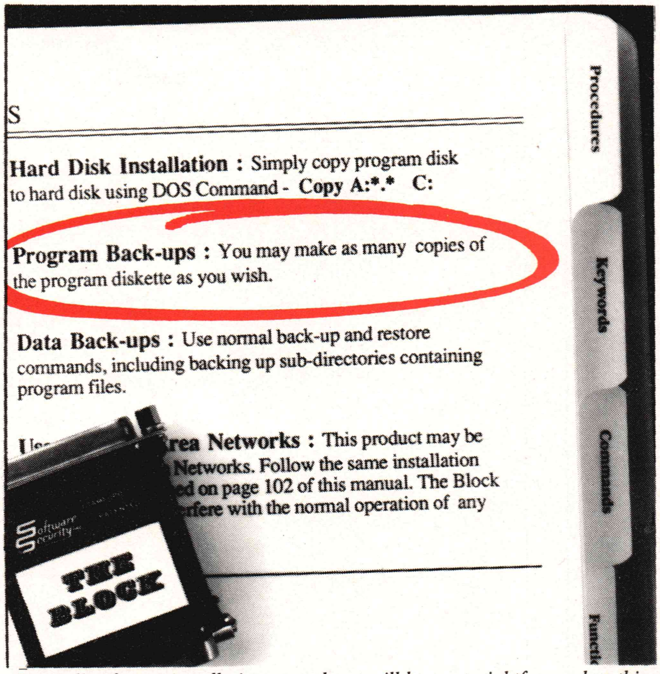
Software protection must be understood to be a distinctively different concept from that commonly referred to as copy protection.

Fundamentally, software protection involves devising a method that prevents unauthorized use of a program, without restricting a legitimate user from making any number of additional copies or preventing program operation via hard disk or LANs.

Logic dictates that magnetic media can no more protect itself from misuse than a padlock can lock itself.

Software protection must reside outside the actual storage media. The technique can then be made as tamper proof as deemed necessary. If one is clever enough, patent law can be brought to bear on the method.

Software protection is at a crossroads and the choices are clear. You can give product away to a segment of the market, or take a stand against the theft of your intellectual property.



Soon all software installation procedures will be as straightforward as this. The only difference will be whether you include the option to steal your product or not.

"...giving your software away is fine..."

We strongly believe that giving your software away is fine, if you make the decision to do so. However, if the public's sense of ethics is determining company policy, then you are no longer in control.

We have patented a device that protects your software while allowing unlimited archival copies and uninhibited use of hard disks and LANs. The name of this product is The BLOCK™.

The BLOCK is the only patented method we know of to protect your investment. It answers all the complaints of reasonable people concerning software protection.

In reality, the only people who could object are those who would like the option of stealing your company's product.

"...eliminating the rationale for copy-busting..."

Since The BLOCK allows a user to make unlimited archival copies the rationale for copy-busting programs is eliminated.

The BLOCK is fully protected by federal patent law rather than the less effective copyright statutes. The law clearly prohibits the production of work-alike devices to replace The BLOCK.

The BLOCK attaches to any communications port of virtually any microcomputer. It comes with a unique customer product number programmed into the circuit.

The BLOCK is transparent to any device attached to the port. Once it is in place users are essentially unaware of its presence. The BLOCK may be daisy-chained to provide security for more than one software package.

Each software developer devises their own procedure for accessing The BLOCK to confirm a legitimate user. If it is not present, then the program can take appropriate action.

"...possibilities... limited only by your imagination..."

The elegance of The BLOCK lies in its simplicity. Once you understand the principle of The BLOCK, hundreds of possibilities will manifest themselves, limited only by your imagination.

Your efforts, investments and intellectual property belong to you, and you have an obligation to protect them. Let us help you safeguard what's rightfully yours. Call today for our brochure, or a demo unit."

Software Security inc.

870 High Ridge Road Stamford, Connecticut 06905

203 329 8870

Listing Twenty-six *(Listing continued)*

```
            p = skipto(' ', rstr , Esc );
            if( *p )
                    *p++ = '\0' ;

            fp->smac[0] = rstr[0] ;
            fp->smac[1] = rstr[1] ;
            fp->smac[2] = '\0';

            p = skipspace( p, Esc );

            fp->emac[0] =  *p                  ? *p++ : '\0' ;
            fp->emac[1] = (*p && *p != ' ') ? *p++ : '\0' ;
            fp->emac[2] = '\0';

            p = skipspace(  p, Esc );   /* and then go to following word */

            fp->left = malloc(  MAX_CHARS_IN_FONT +
                                strlen(Right_str) +
                                strlen(Left_str)   + 2 );
            if( !fp->left )
            {
                err(".df: Not enough memory for width tables\n");
                return;
            }

            fp->right   = cpy( fp->left,  Left_str  ) + 1;
            fp->widths = cpy( fp->right, Right_str ) + 1;

            memset( fp->widths, 1, MAX_CHARS_IN_FONT );

            if( *p )
            {
                if( !(stream = fopen(p,"rb")) )
                    err(".df...%s, Can't open file\n", p);
                else
                {
                    p = fp->widths;
                    i = MAX_CHARS_IN_FONT;

                    while( fscanf(stream, "%d", p) == 1 && --i >= 0 )
                        p++;

                    fclose( stream );
                }
            }
        }
    }
}
/*---------------------------------------------------------*/

di(lstr)
char    *lstr;
{
        /* .di xx        -- divert output to macro xx.
         *                  terminate the diversion with a .di
         *                  or .da (see) without an argument.
         */

        if( *lstr )
                dcreate( lstr );
        else
                endiv();
}

/*---------------------------------------------------------*/

ds(lstr, rstr)
char    *lstr, *rstr;
{
        /* .ds xx str    -- define string xx to hold the
         *                  indicated string. If the string
         *        exists, it is deleted. See also: .as
         */

        if( *lstr )
                screate( lstr, rstr);
        else

                err("Missing string name\n");
}
/*---------------------------------------------------------*/

dt( num, str, offset)
char    *str;
int     num, offset;
{
        /*   .dt [+-]N xx  Set a diversion trap that will
         *                 be sprung after N lines have been
         *   processed in the current diversion. Only one
         *   diversion trap may be active.
         */

        if( !Isdiv )
        {
                err("No diversion currently active\n");
                return;
        }

        if( !num || !*str )          /* Clear existing trap */
        {
                Divtrap    = -1;
                Dtrap_name[0] = 0;
        }
        else if( num >= VERT )  /* Set a diversion trap    */
        {
                Dtrap_name[0] = str[0];
                Dtrap_name[1] = str[1];
```

(continued on page 74)

DAN BRICKLIN'S DEMO PROGRAM ONLY $74.95

Read what they're saying about this popular program for prototyping and demo-making:

"A winner right out of the starting gate. After you use DEMO once, you'll wonder how you got along without it."
—PC Magazine

"Everybody who writes software, either commercially or for in-house applications, should immediately order a copy. Period. No exceptions."
—Soft · letter

Product of the Month
—PC Tech Journal

Thousands of developers and most of the largest and best known software companies are using this program. You can, too. Act now!

NEW TUTORIAL! JUST $49.95

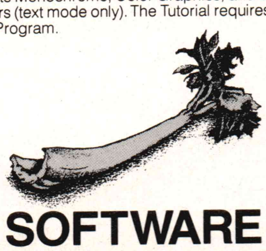
The perfect companion to the Demo Program. The Tutorial helps you learn the ins and outs of its basic and advanced features. Complete with a 96 page manual containing step-by-step instructions, diskette, and function key template.

ORDER NOW!

1-800-CALL-800 x8088

VISA · MasterCard · AMERICAN EXPRESS

Use 800-number for orders only.
Questions, special shipping, etc., call **617-332-2240.**
No Purchase Orders. Massachusetts residents add 5% sales tax. Outside of the U.S.A., add $15.00.
Requires 256K IBM PC/Compatible, DOS 2.0 or later. Supports Monochrome, Color Graphics, and EGA Adapters (text mode only). The Tutorial requires the Demo Program.



SOFTWARE GARDEN, INC.

Dept. D
P.O. Box 373, Newton Highlands, MA 02161

Now available for XENIX

C++

Introducing The Most Important Programming Development Since The Introduction Of C...

ADVANTAGE C++ For MS/PC-DOS, Exclusively From LIFEBOAT.

They say you can't be all things to all people. But Lifeboat's **ADVANTAGE C++** proves that you can be! This exciting new product, developed by AT&T, represents a major programming breakthrough.

ADVANTAGE C++

- Opens the door to object-oriented programming.
- Allows programs with greater resilience, fewer bugs.
- Lets you write reliable, reusable code that is easier to understand.
- Includes many enhancements to C, yet maintains full compatibility with existing C programs.
- Is the key to developing large and sophisticated programs more productively.
- Has all the benefits of C, without its limitations.

ADVANTAGE C++ is now available for the most popular C compilers, Lattice C and Microsoft C.

Why be limited to just C ... when you can have all these pluses!

To order or obtain a complete technical specification sheet call:
1-800-847-7078
In NY: 914-332-1875.
55 South Broadway Tarrytown N.Y. 10591

LIFEBOAT
The Full-Service Source for Programming Software.

CIRCLE 118 ON READER SERVICE CARD

FULL AT&T C++ for half the price of our competitors!

Guidelines announces its port of **version 1.1** of AT&T's C++ translator. As an object-oriented language, C++ includes: classes, inheritance, member functions, constructors and destructors, data hiding, and data abstraction. 'Object-oriented' means that C++ code is more readable, more reliable and more reusable. And that means faster development, easier maintenance, and the ability to handle more complex projects. C++ is **Bell Labs' answer to Ada and Modula 2**. C++ will more than pay for itself in saved development time on your next project.

C++

from GUIDELINES for the IBM PC: $195

Requires IBM PC/XT/AT or compatible with 640K and a hard disk.
Note: C++ is a *translator*, and requires the use of Microsoft C 3.0 or later.

Here is what you get for $195:

- The full AT&T v1.1 C++ translator.
- Libraries for stream I/O and complex math.
- "The C++ Programming Language", the definitive 327-page tutorial and description by Bjarne Stroustrup, designer of C++.
- Sample programs written in C++.
- Installation guide and documentation.
- 30 day money back guarantee.

To order:
send check or money order to:

GUIDELINES SOFTWARE
P.O. Box 749
Orinda, CA 94563

To order with Visa or MC, phone (415) 254-9393.
(CA residents add 6% tax.)

C++ is ported to the PC by Guidelines under license from AT&T.
Call or write for a free C++ information package.

CIRCLE 351 ON READER SERVICE CARD

Listing Twenty-six *(Listing continued)*

```
                setnum( &Divtrap , num, offset );
        }
        else
                err("Passed diversion trap when trap set\n");
}

/*------------------------------------------------------------*/

ec( lstr )
char    *lstr;
{
        /* .ec c          -- Change escape character from \
         *                   to c. Use \ if c is missing.
         */

        Esc = *lstr ? *lstr : '\\' ;
}

/*------------------------------------------------------------*/

el( str)
char    *str;
{
        /* .el    -- else clause part of .ie. This command
         *           is normally processed as part of the .ie
         *           command. If we get here, there is no
         *           corresponding .ie statement.
         *
         *           This command is not inhibitable
         */

        err(".el not associated with .ie\n");
        return 0;
}

/*------------------------------------------------------------*/

em( str )
char    *str;
{
        /* .em xx  --    Define xx as the end macro, executed
         *               after all output has been processed
         */

        Endm = strsave( str );
}

/*------------------------------------------------------------*/

eo(str)
char    *str;
{
        /* .eo    -- NOT AN NROFF COMMAND --
         *
         *          Disable the escape mechanism entirely. It
         *          can be restored again with a .ec command.
         */

        Esc = -1 ;
}

/*------------------------------------------------------------*/

ev( str )
char    *str;
{
        /* .ev N          -- MODIFIED NROFF COMMAND --
         *
         *          This command pushes various commonly used
         *          variables on an environment stack. Nroff
         *          supports several environments and the shell
         *          supports only one. If an argument is present,
         *          the current environment is save. If no
         *          argument is present, a previously saved
         *          environment is popped from the stack. See
         *          push_env and the definition of the
         *          environment strucutre (both in nrmsc.c)
         *          for more information. The stack is five
         *          environments deep.
         */

        if( *str )
                push_env();
        else
                pop_env();
}

/*------------------------------------------------------------*/

ex()
{
        /* .ex    -- exit back to the operating sytem just.
         *          as if input had ended.
         */

        Quit = 1;
}

/*------------------------------------------------------------*/

fi( str, dobreak )
char *str;
{
        /* .fi    --        enable line filling
         */

        if( dobreak )
                brk();
```


(continued on page 78)

Turn the page for our latest advertised price list.

ai - arity products

Arity Combination Package	1095	979
Expert System Development Pkg	295	229
File Interchange Toolkit	50	44
PROLOG Compiler & Interpreter	650	569
Screen Design Toolkit	50	44
SQL Development Package	295	229
Arity PROLOG Interpreter	295	229
Arity Standard Prolog	95	77

ai - expert systems

1st-CLASS *by Programs in Motion*	495	399
AutoIntelligence *by IntelligenceWare*	990	739
ExpertEDGE Advanced *by Human Edge*	2500	CALL
ExpertEDGE Professional *by Human Edge*	5000	CALL
Experteach II *by IntelligenceWare*	475	339
EXSYS *Development Software by EXSYS*	395	309
EXSYS Runtime System	600	469
Insight 1 *by Level Five Research*	95	75
Insight 2+ *by Level Five Research*	485	379
Intelligence/Compiler *IntelligenceWare*	990	739
Logic-Line Series 1 *by Thunderstone*	90	85
Logic-Line Series 2 *by Thunderstone*	125	115
Logic-Line Series 3 *by Thunderstone*	150	139

ai - lisp language

GCLISP *Golden Common LISP by Gold Hill*	495	CALL
GCLISP 286 Developer *by Gold Hill*	1190	CALL
IQCLISP *by Integral Quality* New	300	CALL
IQLISP *by Integral Quality* New	270	CALL
Microsoft LISP *Common LISP*	250	149
QNIAL *Combines LISP & APL by NIAL Systems*	375	339
TransLISP *from Solution Systems*	95	CALL
TransLISP PLUS *from Solution Systems*	195	CALL

ai - prolog language

APT *Active Prolog Tutor from Solution Systems*	65	CALL
LPA microPROLOG *All Varieties*	CALL	CALL
MPROLOG Language Primer *LOGICWARE*	50	45
MPROLOG P500 *by LOGICWARE*	495	395
MPROLOG P550 *by LOGICWARE*	220	175
Prolog-86 *from Solution Systems*	125	CALL
Prolog-86 Plus *from Solution Systems*	250	CALL
Turbo PROLOG *by Borland Intl*	100	63
Turbo PROLOG Toolbox *by Borland Intl*	100	64

ai - smalltalk language

Smalltalk/V *by Digitalk*	99	84
EGA Color Option	49	45
Goodies Diskette	49	45
Smalltalk/Comm	49	42

ai - texas instruments

PC Scheme *Lisp*	95	84
Personal Consultant Easy	495	435
Personal Consultant Plus	2950	2589
Personal Consultant Runtime	95	84

apl language

APL*PLUS/PC *by STSC*	595	424
APL*PLUS/PC Spreadsheet Mgr *by STSC*	195	139
APL*PLUS/PC Tools Vol 1 *by STSC*	295	199
APL*PLUS/PC Tools Vol 2 *by STSC*	85	58
Financial/Statistical Library *by STSC*	275	189
Pocket APL *by STSC*	95	69
STATGRAPHICS *by STSC*	795	579

assembly language

386 ASM/LINK *Cross Asm by Phar Lap*	495	389
8088 Assembler *w/Z-80 Translator by 2500 AD*	100	89
ASMLIB *Function Library by BC Assoc*	149	125
asmTREE *B-Tree Dev System by BC Assoc*	395	339
Cross Assemblers *Various by 2500 AD*	CALL	CALL
Microsoft Macro Assembler	150	93
Norton Utilities *by Peter Norton*	100	59
screenplay *by Flexus*	100	79
Turbo EDITASM *by Speedware*	99	84
Uniware Cross Assemblers *Various by SDS*	295	249
Visible Computer: 8088 *Software Masters*	80	64

basic language

87 QB Pak *by Hauppauge*	69	59
87 Software Pak *by Hauppauge*	180	149
EXIM Services Toolkit *by EXIM*	50	45
Finally *by Komputerwerks*	99	85
Inside Track *from Micro Help*	65	51
MACH 2 *by Micro Help*	69	55
MACH 2 *for Turbo BASIC by Micro Help* New	69	55
Microsoft QuickBASIC Compiler	99	63
Peeks 'n Pokes *from MicroHelp*	45	37
Professional BASIC *by Morgan*	99	68
8087 Math Support	50	42
QuickPak *by Crescent Software* New	69	59
Scientific Subroutine Library *by Peerless*	125	99
Stay-Res *by MicroHelp*	95	73
True Basic *w/Run-time* New Version	245	179
True Basic New Version	150	97
Run-time Module	150	97
Various Utilities	50	41
Turbo BASIC Compiler *by Borland Intl*	100	64

blaise products

ASYNCH MANAGER *Specify C or Pascal*	175	119
C TOOLS PLUS	175	119
EXEC *Program Chainer*	95	73
LIGHT TOOLS *for Datalight C*	100	78
PASCAL TOOLS	125	94
PASCAL TOOLS 2	100	74
PASCAL TOOLS & TOOLS 2	175	119
RUNOFF *Text Formatter*	50	43
TURBO ASYNCH PLUS	100	78
TURBO POWER TOOLS PLUS	100	78
VIEW MANAGER *Specify C or Pascal*	275	179

borland products

EUREKA *Equation Solver*	100	64
Reflex & Reflex Workshop	200	128
Reflex *Data Base System*	150	89
Reflex Workshop	70	45
Sidekick & Traveling Sidekick	125	85
Sidekick	85	57
Traveling Sidekick	70	45
Superkey New Version	100	64
Turbo BASIC Compiler	100	64
Turbo C Compiler New	100	64
Turbo Database Toolbox	70	41
Turbo Editor Toolbox	70	41
Turbo Gameworks Toolbox	70	41
Turbo Graphix Toolbox	70	41
Turbo Jumbo Pack *Combination Package* New	300	219
Turbo Lightning	100	64
Turbo PASCAL Numerical Methods Toolbox	100	64
Turbo PASCAL and Tutor	125	85
Turbo PASCAL	100	64
Turbo Tutor	40	24
Turbo PROLOG Compiler	100	63
Turbo PROLOG Toolbox	100	64
Word Wizard	70	47
Word Wizard and Turbo Lightning	150	94

c++

C++ *by Guidelines w/version 1.1 kernel*	195	172
PforCe++ *Function Library by Phoenix* Sale	395	209

c compilers

68000/10/20 Cross Compiler *by SDS*	595	CALL
C86PLUS *by Computer Innovations*	497	CALL
Datalight C Compiler *Small Model*	60	43
Datalight Developer Kit	99	74
Datalight Optimum-C	139	109
DeSmet C *w/Debugger & Large Case*	209	184
DeSmet C *w/Debugger Only*	159	138
Eco-C *Development System by Ecosoft*	125	83
Lattice C Compiler *from Lattice*	500	265
Mark Williams Let's C Combo Pack	125	99
Let's C *Compiler*	75	54
csd *Source Level Debugger*	75	54
Microsoft C *with CodeView*	450	269
Turbo C Compiler *by Borland Intl* New	100	64

c interpreters

C-terp *by Gimpel, Specify compiler*	300	219
C Trainer *with Book by Catalytix*	122	87
Instant C *by Rational Systems*	500	369
Introducing C *by Computer Innovations*	125	CALL
Run/C *by Age of Reason*	120	79
Run/C Professional *by Age of Reason*	250	157

c utilities

C to dBase *by Computer Innovations*	150	CALL
c-tree & r-tree Combo *by FairCom*	650	519
c-tree *ISAM File Manager*	395	315
r-tree *Report Generator*	295	239
C Windows *by Syscom*	100	85
C Wings *by Syscom*	50	43
CI ROMPac *by Computer Innovations*	195	CALL
dBx *dBase to C Translator by Desktop AI*	350	299
with Library Source Code	550	349
Various Support Utilities	CALL	CALL
Entelekon Products	CALL	CALL
Flash-up Windows *by Software Bottling*	90	78
GraphiC *Color version by Sci Endeavors*	350	282
GraphiC *Mono version by Sci Endeavors*	280	209
GRAFLIB *by Sutrasoft*	175	159
HALO *Graphics by Media Cybernetics*	300	205
HALO Development Pkg *for Microsoft*	595	389
The HAMMER *by OES Systems*	195	129
PANEL *Forms Management by Roundhill*	295	CALL
PANEL Plus *by Roundhill* New	495	CALL
PC Lint *by Gimpel Software*	139	99
PLOTHI *by Sutrasoft*	175	159
PLOTHP *by Sutrasoft*	175	159
Professional C Windows *by Washburn* New	89	79
Scientific Subroutine Library *by Peerless*	175	128
screenplay *for C by Flexus*	175	129
Vitamin C *by Creative Programming*	225	158
VC Screen *Forms Designer*	100	79
Zview *by Data Mgmt Consultants* Special Offer	245	CALL

cobol language

COBOLspII *by Flexus* New	395	329
EASY SCREEN *by Retail Mgmt Systems* New	225	199
FPLIB *for Realia COBOL by BC Associates* New	149	129
Micro Focus COBOL *See Micro Focus Section*		
Microsoft COBOL *See Microsoft Section*		
Realia COBOL	995	783
RealCICS	995	783
RealMENU New	150	119
RM/COBOL *by Ryan-McFarland*	950	639
RM/COBOL 85 *by Ryan-McFarland*	1250	895
screenplay *for COBOL by Flexus*	175	129

debuggers & profilers

386 DEBUG *Cross Debugger by Phar Lap*	195	129
Advanced Trace-86 *by Morgan Computing*	175	115
CI Probe *by Computer Innovations*	225	CALL
Codesifter *Profiler by David Smith*	119	85
Codesmith-86 *by Visual Age*	145	98
DSD87 *by Soft Advances*	125	99
MiniProbe *by Atron*	395	CALL
Periscope I *with Board by Periscope*	345	289
Periscope II *with NMI Breakout Switch*	175	139
Periscope II-X *Software only*	145	105
Periscope III *w/Advanced Board* New	995	CALL
The PROFILER *with Source Code by DWB*	125	89
The WATCHER *Profiler by Stony Brook*	60	51

dos utilities

Command Plus *by ESP Software* New	80	69
FANSI-CONSOLE *by Hersey Micro*	75	62
MKS Toolkit *with vi Editor by MKS*	139	99
Norton Commander *by Peter Norton*	75	55
Scroll & Recall *by Opt-Tech Data*	69	59
Taskview *by Sunny Hill Software*	80	56

CIRCLE 129 ON READER SERVICE CARD

essential products

Special Offers Available. Call for Details.

C Essentials *by Essential Software*	100	75
C Utility Library	185	119
Essential Comm Library *with Debugger*	250	189
Essential Comm Library *Software Only*	185	125
Breakout Debugger *Any language*	125	89
Essential Graphics	250	183

forth language

CFORTH *Native Code Compiler by LMI*	300	229
Forth/83 Metacompiler *Specify Target*	750	599
PC/Forth *by Laboratory Microsystems*	150	109
PC/Forth+ *by Laboratory Microsystems*	250	195
Advanced Color Graphics Support	100	74
Enhanced Graphics Support	200	148
Intel 8087 Support	100	74
Interactive Symbolic Debugger	100	74
Native Code Optimizer	200	148
Software Floating Point	100	74
UR/Forth *by LMI*	350	275
Object Module Libraries	500	395
Source Code License	1500	995

fortran language

50 MORE: FORTRAN *by Peerless Engr*	125	95
ACS Time Series *Alpha Computer Service*	495	389
Essential Graphics *by Essential Software*	250	183
For-Winds *Alpha Computer Service*	90	69
Forlib-Plus *Alpha Computer Service*	70	44
FORTLIB *by Sutrasoft*	125	109
FORTRAN Addendum *by Impulse Engr*	95	85
FORTRAN Addenda *by Impulse Engr*	165	138
GRAFLIB *by Sutrasoft*	175	159
HALO *by Media Cybernetics*	300	205
I/O PRO *by MEF Environmental*	149	129
Microcompatibles Combo Package	240	215
Grafmatic	135	117
Plotmatic	135	117
Microsoft FORTRAN *w/CodeView*	450	269
No Limit *by MEF Environmental*	129	109
Numerical Analyst *by MAGUS* New	295	CALL
PANEL *by Roundhill Computer Systems*	295	CALL
PLOTHI *by Sutrasoft*	175	159
PLOTHP *by Sutrasoft*	175	159
RM/FORTRAN *Ryan-McFarland*	595	389
RTC PLUS *Fortran to C by Cobalt Blue* New	325	CALL
Scientific Subroutine Lib *by Peerless*	175	128
Statistician *Alpha Computer Service*	295	235
Statlib.GL: *by PSI/Systems* New	295	239
Statlib.TSF: *by PSI/Systems* New	295	239
Strings & Things *Alpha Computer Service*	70	45

greenleaf products

Greenleaf Comm Library	185	125
Greenleaf Data Windows	225	157
with Source Code	450	289
Greenleaf Functions	185	125

help utilities

HELP/Control *by MDS*	125	99
On-line Help *from Opt-Tech* New Version	149	99
SoftScreen/HELP *by Dialectic Systems*	195	149

lattice products

Lattice C Compiler *from Lattice*	500	265
with Library Source Code	900	495
C Cross Reference Generator	50	37
with Source Code	200	139
C-Food Smorgasbord *Function Library*	150	95
with Source Code	300	179
C-Sprite *Source Level Debugger*	175	128
Curses *Screen Manager*	125	88
with Source Code	250	174
dBC II *Specify dBC II or dBC III*	250	169
with Source Code	500	356
dBC III Plus New	750	CALL
with Source Code New	1500	CALL
LMK *Make Facility*	195	138
RPG II Combo *All three items below*	1100	875
RPG II Compiler *No Royalties*	750	625
RPG II SEU *Screen Entry Utility*	250	199
RPG II Sort/Merge	250	199
RPG II Screen Design Aid Utility	350	309
SecretDisk *File Encryption Utility*	120	88
SideTalk *Resident Communications*	120	88
SSP/PC *Scientific Subroutine Library*	350	269
Text Management Utilities	120	88
TopView Toolbasket *Function Library*	250	178
with Source Code	500	356

metagraphics products

LightWINDOWS/C *for Datalight C* New	95	79
MetaWINDOWS *No Royalties*	185	109
MetaFONTS	80	58
MetaWINDOWS/Plus	235	184
MetaFONTS/Plus	235	184
TurboWINDOW *Graphics/Windows for Turbo Pascal*	80	58

micro focus products

Micro Focus COBOL Workbench	4000	3379
Micro Focus Level II COBOL Special Pricing	1500	549
COGRAPHICS	250	199
COMATH	200	159
FORMS-2	300	259
Level II Animator	895	769
Level II SOURCEWRITER	995	849
Micro Focus Personal COBOL New	149	CALL
Micro Focus Professional COBOL	3000	2549
Multi-user Runtime *for PC Network*	500	399

microport products

System V/AT *by Microport Systems* New Version	549	475
Runtime System *(Operating System)*	199	189
Software Development System	249	235
Text Preparation System	199	189
User Upgrade *3 to Unlimited Users*	249	235

microsoft products

Microsoft BASIC Interpreter *for XENIX*	350	209
Microsoft C *with CodeView*	450	269
Microsoft COBOL *w/COBOL Tools* ...... *New Version*	700	429
for XENIX	995	609
Microsoft FORTRAN *w/CodeView*	450	269
for XENIX	695	419
Microsoft Learning DOS	50	36
Microsoft LISP *Common LISP*	250	149
Microsoft MACH 10 *w/Mouse & Windows*	549	369
Microsoft MACH 10 *Board only*	399	279
Microsoft Macro Assembler	150	93
Microsoft Mouse *Bus Version*	175	114
Microsoft Mouse *Serial Version*	195	124
Microsoft muMath *Includes muSIMP*	300	179
Microsoft Pascal Compiler	300	179
for XENIX	695	419
Microsoft QuickBASIC Compiler	99	63
Microsoft Sort	195	125
Microsoft Windows	99	63
Microsoft Windows Development Kit	500	299

modula-2 language

IOTools *by Rhodes Associates* ...... *New*	80	69
with Source Code ...... *New*	950	CALL
MODULA-2 Apprentice Pkg by LOGITECH ...... *New*	99	79
MODULA-2 Magic Pkg by LOGITECH ...... *New*	99	79
MODULA-2 ROM Pkg & Cross RT Debugger .... *New*	299	239
MODULA-2 Window Pkg by LOGITECH ...... *New*	49	39
MODULA-2 Wizard's Pkg by LOGITECH ...... *New*	199	159
REPERTOIRE *for MODULA-2 by PMI*	89	79
Object Code Only	19	15

mouse products

LOGIMOUSE BUS *with PLUS Pkg by LOGITECH*	119	98
with PLUS & PC Paintbrush	169	134
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179
LOGIMOUSE C7 *with PLUS Pkg, Specify Connector*	119	98
with PLUS & PC Paintbrush	169	134
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179

other languages

CCS MUMPS *Single-User by MGlobal*	60	49
CCS MUMPS *Single-User/Multi-Tasking*	150	129
CCS MUMPS *Multi-User*	450	359
Janus/ADA C Pak *by R&R Software*	95	84
Janus/ADA D Pak *by R&R Software*	900	769
Janus/ADA ED Pak *by R&R Software* ...... *New*	395	CALL
Marshal Pascal *by Marshal Language Systems* ... *New*	189	155
Personal REXX *by Mansfield Software*	125	99
SNOBOL4+ *by Catspaw*	95	79

other products

Dan Bricklin's Demo Pgm *Software Garden*	75	57
Dan Bricklin's Demo Tutorial ...... *New*	50	45
Disk Optimizer *by Softlogic Systems* ...... *New*	60	55
FASTBACK *by 5th Generation Systems*	179	133
Instant Replay *by Nostradamus*	90	79
Net-Tools *by BC Associates* ...... *New*	149	129
OPT-Tech Sort *by Opt-Tech Data Proc*	149	99
Screen Machine *by MicroHelp* ...... *New*	79	59
VTEK *Term Emulator by Sci Endeavors*	150	129

phoenix products

Pasm86 *Macro Assembler Version 2.0*	195	108
Pdisk *Hard Disk & Backup Utility*	145	88
Pfantasy Pac *Phoenix Combo* ...... *Sale*	1295	799
Pfinish *Execution Profiler* ...... *Sale*	395	209
Pfix86plus *Symbolic Debugger* ...... *Sale*	395	209
PforCe *Comprehensive C Library* ...... *Sale*	395	209
PforCe++ *Library for Guidelines C++* ...... *Sale*	395	209
Plink86plus *Overlay Linker* ...... *Sale*	495	279
Pmaker *Make Utility*	125	78
Pmate *Macro Text Editor*	195	108
Pre-C *Lint Utility*	295	154
Ptel *Binary File Transfer Program*	195	108

polytron products

PolyBoost *The Software Accelerator*	80	64
PolyLibrarian *Library Manager*	99	73
PolyLibrarian II *Library Manager*	149	109
PolyMake *UNIX-like Make Facility*	149	109
PolyShell	149	109
Polytron C Beautifier	50	42
Polytron C Library I	99	72
Polytron PowerCom *Communications*	139	105
PolyWindows Products *All Varieties*	CALL	CALL
PolyXREF *Complete Cross Ref Utility*	219	169
PolyXREF *One language only*	129	99
PVCS Corporate *Version Control System* ...... *New*	395	309
PVCS Personal ...... *New*	149	109

program mgmt utilities

Compact Source Print *by Aldebaran*	55	44
Interactive EASYFLOW *by Haventree*	150	125
PrintQ *by Software Directions*	89	84
Quilt Computing Combo Package	199	159
QMake *Program Rebuild Utility*	99	79
SRMS *Software Revision Mgmt Sys*	125	109
Source Print *by Aldebaran Labs*	75	59
TLIB *by Burton Systems Software*	100	89
Tree Diagrammer *by Aldebaran Labs*	55	49

raima products

dbQUERY *Single-User Query Utility*	195	129
Single-User with Source Code	495	389
Multi-User	495	389
Multi-User with Source Code	990	799
dbVISTA *Single-User DBMS*	195	129
Single-User with Source Code	495	389
Multi-User	495	389
Multi-User with Source Code	990	799

sco products

Complete XENIX System V *by SCO*	1295	994
Development System	595	499
Operating System *Specify XT or AT*	595	499
Text Processing Package	195	144
Networks for XENIX *by SCO*	595	495
SCO Professional *Lotus clone for XENIX*	795	595

softcraft products

Btrieve *ISAM Mgr with No Royalties*	245	184
Xtrieve *Query Utility*	245	184
Report Option *for Xtrieve*	145	99
Btrieve/N *for Networks*	595	454
Xtrieve/N	595	454
Report Option/N *for Xtrieve/N*	345	269

solution systems products

APT *Active Prolog Tutor*	65	CALL
Brief & dBrief Combo	250	CALL
Brief *Programmer's Text Editor*	195	CALL
dBrief *Customizes Brief for dBase III*	95	CALL
C Screen Editor	75	CALL
C ToolSet	95	CALL
Faster C	95	CALL
Prolog-86	125	CALL
Prolog-86 Plus	250	CALL
Security Library	125	CALL
with Source Code	250	CALL
TransLISP	95	CALL
TransLISP PLUS	195	CALL
ZAP *Communications*	95	CALL

text editors

Brief *from Solution Systems*	195	CALL
Epsilon *Emacs-like editor by Lugaru*	195	147
KEDIT *by Mansfield Software*	125	98
Micro/SPF *by Phaser Systems*	175	139
PC/VI *by Custom Software Systems*	149	99
SPF/PC *by Command Technology Corp* ... *New Version*	245	175
Vedit *by CompuView*	150	98
Vedit Plus *by CompuView*	185	128

turbo pascal utilities

ALICE *Interpreter by Software Channels*	95	66
DOS/BIOS & Mouse Tools *by Quinn-Curtis* ...... *New*	75	67
Flash-up Windows *by Software Bottling*	90	78
MACH 2 *for Turbo Pascal by Micro Help* ...... *New*	69	55
MetraByte D/A Tools *by Quinn-Curtis* ...... *New*	100	89
Science & Engrg Tools *by Quinn-Curtis* ...... *New*	75	67
Screen Sculptor *by Software Bottling*	125	91
screenplay *for Turbo Pascal by Flexus*	100	79
Speed Screen *by Software Bottling*	35	32
System Builder *by Royal American*	100	CALL
IMPEX *Query Utility*	75	CALL
Report Builder	75	CALL
TDebugPLUS *by TurboPower Software*	60	49
Turbo EXTENDER *by TurboPower Software*	85	64
Turbo Professional *by Sunny Hill*	70	45
TurboHALO *from IMSI*	129	98
TurboPower Utilities *by TurboPower*	95	78
TurboRef *by Gracon Services*	50	45
TURBOsmith *Visual Age Debugger* ...... *Special Price*	69	45

wendin products

Operating System Toolbox ...... *Rebate Offer*	99	75
PCNX *Operating system* ...... *Rebate Offer*	99	75
PCVMS *Similar to VAX/VMS* ...... *Rebate Offer*	99	75
XTC *Text Editor w/Pascal source* ...... *Rebate Offer*	99	75

xenix/unix products

Btrieve *ISAM File Mgr by SoftCraft*	595	454
C-terp *by Gimpel, Specify compiler*	498	379
c-tree *ISAM Mgr by FairCom*	395	315
dbVISTA *See Raima Section*		
dBx *with Library Source by Desktop AI*	550	489
DOSIX *Console Version by Data Basics*	399	CALL
DOSIX *User Version by Data Basics*	199	CALL
Micro Focus Level II Compact COBOL	1000	CALL
Forms-2	400	CALL
Level II ANIMATOR	600	CALL
Microport Products *See Microport Section*		
Microsoft Products *See Microsoft Section*		
PANEL Plus *by Roundhill Computer Systems* ...... *New*	CALL	CALL
REAL-TOOLS *Binary Version by PCT*	149	89
Library Source Version	399	289
Complete Source Version	499	369
RM/COBOL *by Ryan-McFarland*	1250	949
RM/FORTRAN *by Ryan-McFarland*	750	549
SCO Products *See SCO Section*		

LOWEST PRICES
Due to printing lead times, some of our current prices may differ from those shown here. Call for latest pricing.

FREE SHIPPING
Orders within the USA (including Alaska & Hawaii) are shipped FREE via UPS. Express shipping is available at the shipping carrier's standard rate with no rush fees or handling charges. To avoid delays when ordering by mail, please call first to determine the exact cost of express shipping.

CREDIT CARDS
VISA and MasterCard are accepted at no extra cost. Your card is charged when your order is shipped. Mail orders please include credit card expiration date and telephone number.

CODs AND POs
CODs and Purchase Orders are accepted at no extra cost. POs with net 30-day terms are available to qualified US accounts only.

FOREIGN ORDERS
Shipping charges for foreign and Canadian orders are based on the shipping carrier's standard rate. Since rates vary between carriers, please call or write for the exact cost. Foreign orders (except Canada), please include an additional $10 for customs form preparation. All payments must be made with US funds drawn on a US bank. Please include your telephone number when ordering by mail. Due to government regulations, we cannot ship to all countries.

VOLUME ORDERS
Volume orders may qualify for additional discounts. Call us for special pricing.

SOUND ADVICE
Our knowledgeable technical staff can answer technical questions, assist in comparing products and send you detailed product information tailored to your needs.

30-DAY GUARANTEE
Most of our products (excluding books) come with a 30-day documentation evaluation period or a 30-day return guarantee. Please note that some manufacturers restrict us from offering guarantees on their products. Call for more information.

MAIL ORDERS
Please include your telephone number on all mail orders. Be sure to specify computer, operating system and any applicable compiler or hardware interface(s). Send mail orders to:

Programmer's Connection
136 Sunnyside Street
Hartville, OH 44632

CALL TOLL FREE

U S	**800-336-1166**
CANADA	**800-225-1166**
OHIO & ALASKA (Call Collect)	**216-877-3781**
TELEX	**9102406879**
FOREIGN	**216-877-3781**
CUSTOMER SERVICE	**216-877-1110**

Hours: Weekdays 8:30 AM to 8:00 PM EST.

Ohio customers add 6% state sales tax.
Prices are subject to change without notice.

programmer's connection

CIRCLE 129 ON READER SERVICE CARD

Listing Twenty-six *(Listing continued)*

```
        FILL = 1;
}

/*--------------------------------------------------------*/

ft( str )
char *str;
{
        /* .ft F   --   Change font to F at the
         *              beginning of the next
         * input text line. Font changes can also be
         * imbedded with a \fF escape sequence. Note that
         * if font F doesn't exist, the error won't be
         * flagged until the output routines try to process
         * the font change request. F may be a number that
         * was fetched from the \n(.f number register at
         * some earlier time.   .ft 0 is the same as .ft R.
         */

        chgfont( *str );
}

/*--------------------------------------------------------*/

hd( num, str, offset, dobreak, tail )
char    *str;
char    *tail;
{
        /* .hd <left str> N <right str> -- NOT NROFF ---
         *
         * Define strings to send printer cursor left or
         * right by 1/N spaces. The width of a space is
         * taken from the currently active font width
         * table. It will be 1 in the default, non-
         * proportionally spaced font. N determines the
         * minimum resolution for the space between
         * characters in proportional spacing mode.
         */

        static char lstr[81];
        static char rstr[81];

        strncpy( lstr, str , 80 );    lstr[80] = 0;
        strncpy( rstr, tail, 80 );    rstr[80] = 0;

        Left_str  = lstr;
        Right_str = rstr;
        Hs_amt    = num;
}

/*--------------------------------------------------------*/

hy( num )
{
        /* .hy [N]     -- MODIFIED NROFF COMMAND --
         *
         *      Enable hyphenation. N is ignored.
         */

        Hyphenate = 1;
}

/*--------------------------------------------------------*/

id( lstr, rstr )
char    *lstr, *rstr;
{
        /* .id on off    -- NOT AN NROFF COMMAND ---
         *
         * Send "on" to the printer to put it into italics
         * (underline) mode, rstr to take it out. Maximum
         * length of either string is 80 characters. Use
         * \x<two hex digits> to send a control character.
         */

        static char     on[81], off[81] ;
        on[80] = off[80] = 0;

        strncpy( on,  lstr, 80 );
        strncpy( off, rstr, 80 );

        Ul_on  = on ;
        Ul_off = off;
}

/*--------------------------------------------------------*/

doif( expr, action )
char    *expr, *action;
{
        /* Test an expression and do an if statement (or
         * the if part of a .ie. Set Inhibit as appropriate.
         * Modify Inhibit to reflect the expression.
         * We call process if input is inhibited in order
         * to handle nested if's and blocks. Return 1 if
         * input was not inhibited and we executed the
         * expression, otherwise return 0.
         */

        int     rval = 0;

        if( !Inhibit )
        {
            if( startexpr(expr) )
                Inhibit = ! (int) parse( &expr );

            else if( *expr == 'e' )
```

(continued on page 80)

Listing Twenty-six *(Listing continued)*

```
                Inhibit = (PAGE & 1);

            else if( *expr == 'o' )
                Inhibit = !(PAGE & 1);

            else
            {
                Inhibit = 1;
                err("Illegal expression\n");
            }

            rval = 1;
        }

        process( action, Ifilename, 2, Macv );
        return rval;
}

/*------------------------------------------------------*/

iff(lstr, rstr)
char    *lstr, *rstr;
{
        /* .if condition action
         *
         *      Simple if statement (doesn't take an else
         *      clause). The expression parser used to
         *      evaluate the <condition> is more powerful
         *      than NROFF's. Multi-line blocks can be
         *      used by using a .{ as an action. (be sure
         *      to terminate the block with a .}
         *
         *      If input is inhibited we want to process
         *      the tail without modifying the inhibit status
         *      (in case the tail is a .{ command; otherwise
         *      we set inhibit based on the value of the
         *      expression and then process the tail.
         *
         *      This command is not inhibitable
         */

        if( doif(lstr, rstr) )
            Inhibit = 0;

        return 0;
}
/*------------------------------------------------------*/

char    *iselse( str )
char    *str;
{
    /* Used by ie() (below) returns 0 if str doesn't
     * hold a legal .el command, otherwise returns
     * a pointer to just past the 'l'.
     */

    if( !ISCMD( *str++ ))
        return 0;

    while( isspace(*str) )
        str++;

    return( str[0]=='e' && str[1]=='l' ) ? str+2 : 0 ;
}

/*------------------------------------------------------*/

ie(lstr, rstr)
char    *lstr, *rstr;
{
    /* .ie condition action -- MODIFIED NROFF COMMAND
     *
     * if part of an if/else. Is only non-standard in
     * that the expression parser is more powerful
     * than NROFF's
     *
     * This command is not inhibitable
     */

    static  char        line [MAXSTR], lnum;
    int                 did_something;

    /*  Remember the current line number for the
     *  sake of the error message printed when we can't
     *  find an else clause.
     */

    lnum = INLINES ;

    did_something = !Inhibit;
    doif( lstr, rstr );

    if( !getline(line, 0, Ismacro? mgetc: fgetc) || !
                                        (rstr=iselse(line)))
    {
            /*  Complain if the line that should contian
             *  the .el and isn't there.
             */

            err("Missing .el for .ie on line %d\n", lnum );
    }
    else
    {
            if( did_something )
                Inhibit = !Inhibit ;
```

(continued on page 82)

Listing Twenty-six *(Listing continued)*

```
                    /* Intitialize rstr to point just past the
                     * .el, skip any white space and quotes if
                     * necessary. Then call process with the
                     * command tail as its argument.
                     */

                    rstr = skipspace(rstr, Esc);
                    if( *rstr == '"' )
                            *skipto('"', ++rstr, Esc) = 0;

                    process( rstr, Ifilename, 2, Macv );
            }

            if( did_something )
                    Inhibit = 0;

            return 0;
}

/*-----------------------------------------------------------*/

ig( str )
char    *str;
{
        /* .ig -- Ignore input by writing to a dummy macro
        */

        mwrite( (char *)0,  str );
}

/*-----------------------------------------------------------*/

in( num, str, offset, dobreak )
char    *str;
{
        /*  .in [+-] N  -- Change the indent level
        */

        if( dobreak )
                brk();

        setnum( &INDENT, num, offset );
}

/*-----------------------------------------------------------*/

it( num, str, offset )
char    *str;
int     num, offset ;
{
        /* .it [+-]N xx  Input line trap. Spring macro
         *                xx after N lines of input have been
         *  read. Only one input line trap may be active.
         *  A .it destroys a previous trap if on exists.
         */

        if( !num || !*str )
        {
                /* Remove current input trap    */

                Itrap           = -1;
                Itrap_name[0] =  0;
                return;
        }

        setnum( &Itrap, num, offset );

        if( Itrap <= 0 )
        {
                err(".it xx N: N may not be negative\n");
                Itrap = -1;
        }
        else
        {
                Itrap_name[0] = str[0];
                Itrap_name[1] = str[0];
        }
}

/*-----------------------------------------------------------*/

lc( str )
char    *str;
{
        /* .lc C -- Change leader character from . to C
        */

        Leader = *str ;
}

/*-----------------------------------------------------------*/

ll( num, str, offset )
char    *str;
{
        /* .ll [+-]N      --     Change line length to N
        */

        setnum( &LINLEN, num, offset );
}

/*-----------------------------------------------------------*/

ls( num, str, offset )
char    *str;
{
        /* .ls N  --  Change line spacing to N spaces
        */
```

```
        if( num >= 1 )
                setnum( &LSPACE, num, offset );
        else
                err("\".ls N\",  N must be >= 1\n");
}

/*------------------------------------------------------------*/

lt( num, str, offset )
char    *str;
{
        /* .lt [+-]N -- Change the length of a 3-part title
         */

        setnum( &Title_len, num, offset );
}

/*------------------------------------------------------------*/

mc( num, str )
char    *str;
{
        /* .mc str [N] --  ENHANCED NROFF COMMAND --
         *
         *  Print the string, str, N spaces to the right
         *  of the right margin. This differs from nroff,
         *  which uses a single character rather than
         *  a string. If str and N are both missing, the
         *  margin  character is disabled. The string is
         *  limited to 20 characters (including any spaces
         *  implied by N). If N is missing or 0, 2 is used.
         */

        static UCHAR    buf[21];
        UCHAR           *p;

        if( num <= 0 )
                num = 2;

        for( p = buf; --num >= 0 ; *p++ = ' ' )
                ;

        strncpy( p, str, &buf[21] - p );
        Rmarg_str = buf;
}

/*------------------------------------------------------------*/

mf( macro, file )
char    *macro, *file;
{
        /* .mf xx file  -- NOT AN NROFF COMMAND --
         *
         * Copy the contents of the macro xx to the
         * indicated file. The macro may not be in
         * use at the time. This command is particularly
         * useful for saving a collected index that hasn't
         * been sorted yet.
         */

        dump_mac( macro, file );
}

/*------------------------------------------------------------*/

ml( str )
{
        /* .ml str -- NOT AN NROFF COMMAND
         *
         * Like a mc but prints the string at the left
         * margin rather than the right margin. The page
         * offset must be at least as large as the string,
         * which is limited to 21 characters.
         */

        static char  buf[21];

        strncpy( buf, str, 20 );
        Lmarg_str = buf;
}

/*------------------------------------------------------------*/

na( str )
char    *str;
{
        /* .na  --  Turn of adjusting
         */

        Adjusting = 0;
}

/*------------------------------------------------------------*/

nb( str )
char    *str;
{
        /* .nb -- NOT AN NROFF COMMAND --
         *
         * Used in conjuntion with a .nm, will cause blank
         * lines to be numbered as well as nonblank lines.
         * Useful if you're using nr to format listings.
         */

        Nm_blanks = *str ;
}

/*------------------------------------------------------------*/
```

(continued on next page)

Listing Twenty-six *(Listing continued)*

```
ne( num )
{
        /* .ne N  --  If the distance between the current
         *            line and the next output line trap is
         *  less than N, skip forward to the next trap.
         *  The assumption is that the trap will be an
         *  end of page trap.
         */

        register int    i;

        if( (i = TOTRAP) < num )
                prblank( i );
}

/*------------------------------------------------------*/

nf( str, dobreak )
char *str;
{
        /* .nf  --      Disable line filling, flushing the
         *              buffer first.
         */

        if( dobreak )
                brk();

        FILL = 0;
}

/*------------------------------------------------------*/

nh()
{
        /* .nh   --     Turn off hyphenation (that was
         *              turned on with a .hy command).
         */

        Hyphenate = 0;
}

/*------------------------------------------------------*/

nm( str )
char    *str;
{
    /* .NM N M S         -- MODIFIED NROFF COMMAND --
     *
     *  N = first line number
     *  M = only even multiples of M are printed
     *  S = print string after number
     *
     * If you need to change M without changing N, use
     * .nm x M S where x is any non-number. Same goes for
     * .nr x x S.
     *
     * If no arguments are specified turn off numbering
     * but remember current line number etc. Use .nm x
     * to resume where x is any non-numeric argument.
     *
     * Bugs: The arrays gotten from malloc()
     *   for the S argument are never free()ed.
     */

    char                *p;
    extern double       parse();

    if( Nm_on = *str )
    {
        splitfields(&str, &p);          /* Do N argument  */
        if( isdigit(*str) )
            LINE = (int) parse( &str );

        splitfields(&p, &str);          /* Do M argument  */

        if( isdigit(*p) )
            Nm_mult = (int) parse( &p );

        if( *str )                      /* Do S argument  */
        {
            if( !(Nm_str = strsave(str)) )
            {
                err("Can't get enough memory for .nm\n");
                Nm_str = " " ;
            }
        }
    }
}

/*------------------------------------------------------*/

nr( num, str, offset, dobreak, tail )
char    *str;
char    *tail;
{
        /* .nr R [+-]N [[-]M]    -- ENHANCED NROFF COMMAND --
         *
         * create or modify number register R by (to) N. If
         * M is present, it is incremented when invoked
         * with \n+x, \n+(xx, \n-x or \n+(xx. If M is absent,
         * 1 is used. Unlike nroff, .nr, with no arguments,
         * prints all currently defined number registers.
         */

        if( *str == '\0')
                pr_nregs();
        else
```

(continued on page 86)

Listing Twenty-six *(Listing continued)*

```
}               putnreg( str, 0, num, offset, 1, atoi(tail) );

/*------------------------------------------------------------*/

ns( str)
char    *str;
{
        /*  .ns    --    Inhibit line spacing: Dont' print
         *               newlines until some text is
         *   printed or a .bp N (the N is required) or a
         *   .rs is executed
         */

        Nospace = 1;
}

/*------------------------------------------------------------*/

od(lstr, rstr)
{
        /*  .od on off   -- NOT AN NROFF COMMAND --
         *
         *      Define two strings, one to enter overstrike
         *      mode on the printer (on) and a second to
         *      exit (off). Maximum string length is
         *      80 characters.
         */

        static char     on[81], off[81] ;
        on[80] = off[80] = 0;

        strncpy( on,  lstr, 80 );
        strncpy( off, rstr, 80 );

        Os_on  = on ;
        Os_off = off;
}

/*------------------------------------------------------------*/

os( num, str, offset )
{
        /* .os [+-]N     -- NOT AN NROFF COMMAND --
         *
         * Just like .ul except it overstrikes the next
         * N input lines rather than underlining them.
         */

        setnum( &Num_os, num, offset);
}

/*------------------------------------------------------------*/

ou(str)
char    *str ;
{
    /* .ou str   -- NOT AN NROFF COMMAND --
     *
     *   Output string directly to the current output,
     *   without going through the normal text processing
     *   mechanism.  Line number, adjusting, etc. will
     *   not be affected. This command for sending control
     *   sequences directly to the printer (ie. for
     *   initializations etc. Use \x<2 hex digits> to send
     *   non-printing characters. The top bit of the
     *   character is trimmed off before transmitting, so
     *   you can send an ASCII null as a "\x80". Also note
     *   that the -c flag (which causes control characters
     *   to be printed in readable form) has affect on the
     *   output of this command.
     */

    ots( str );
}

/*------------------------------------------------------------*/

pc( lstr )
char    *lstr;
{
        /* .pc C   -- Change the character used to indicate
         *      a page number in a 3-part title (.tl) from
         *      % to C.
         */

        Page_ch = *lstr ;
}

/*------------------------------------------------------------*/

pl( num, str, offset )
char    *str;
int     num;
{
        /* .pl [+-]N   -- Set page length to N
         */

        setnum( &PGLEN, num, offset);
}
/*------------------------------------------------------------*/

po( num, str, offset )
char    *str;
{
        /* .po [+-]N   -- Set page offset to N
         */
```

(continued on page 90)

THE PROGRAMMER'S SHOP

provides complete information, advice, guarantees and every product for Microcomputer Programming.

Favorite Phoenix Tools

Phoenix provides productivity for every programmer. From object oriented C tools to an overlay linker, plus a debugger, a powerful, fast assembler, and a high-performance editor — Phoenix has it all. **Call one of our specialists TODAY.**

Order before May 31, 1987 and mention this ad for these SPECIAL PRICES:

	List	Normal	SPECIAL
Pasm Assembler-improved	$195	$115	**$105**
Pfix + Debugger	$395	$229	**$209**
Pforce Library	$395	$229	**$209**
Pforce + + Library	$395	$279	**$209**
Plink 86 PLUS	$495	$319	**$279**
Pmate Editor	$195	$115	**$105**

C Support-Systems

Advantage C + + - object oriented	PC	Call
Basic-C Library by C Source	PC	$139
C Sharp - **realtime,** tasks.	PC	$600
C ToolSet - DIFF, xref, source	MS	$ 95
The HAMMER by OES Systems	PC	$139
Lattice Text Utilities	MS	$ 89
Multi-C - multitasking	PC	$149
PC LINT-Checker. Amiga $89	MS	$ 99
Quickshell - script compiler	PC	$349
Pfantasy Pac - by Phoenix	PC	$849
Pre-C - Lint-Like	MS	$155
Programmer's Extender, Vol. I	MAC	$ 79
Sapiens V8 - 8M workspace	PC	$269
SECURITY LIB-add encrypt to MS C, C86 programs. Source $229	PC	$115
Time Slicer - R/T	PC	$265

C-Screens, Windows, Graphics

C Power Windows by Entelekon	PC	$109
dBASE Graphics for C	PC	$ 69
C-Scape - capture Dan Bricklin	PC	$179
Curses by Lattice	PC	$ 89
ESSENTIAL GRAPHICS - fast	PC	$195
GraphiC - mono version	PC	$209
GraphiC - new color version	PC	$285
Greenleaf Data Window	PC	$159
w/source	PC	$319
Multi-Windows - use w/ Multi-C	PC	$295
Screen Ace Form Master	PC	$195
Vitamin C - screen I/O	PC	$199
Windows for C - fast	PC	$149
Windows for Data - validation	PC	$239
View Manager - by Blaise	PC	$189
ZView - screen generator	MS	$175

Debuggers

386 Debug - by Phar Lap	PC	$129
Breakout - by Essential	PC	$ 89
CODESMITH - visual	PC	$ 99
C SPRITE - data structures	PC	$129
DSD87 - by Soft Advances	PC	$ 79
Periscope I - List: $345	PC	Call
Periscope II - List: $175	PC	Call
Periscope II-X - List: $145	PC	Call
Pfix-86 Plus - by Phoenix	PC	$229
Showcase - test software	PC	$135
SoftProbe II - by Systems & Software, embedded systems	PC	$695

Fortran & Supporting

50:More FORTRAN	PC	$ 99
ACS Time Series	MS	$399
Forlib + by Alpha	MS	$ 59
MACFortran by Microsoft	MAC	$229
MS Fortran - 4.0, full 77	MS	$299
No Limit - Fortran Scientific	PC	$115
PC-Fortran Tools - xref, pprint,	PC	$179
RM/Fortran		Call
Scientific Subroutines - Matrix	MS	$139

Multilanguage Support

BTRIEVE ISAM	MS	$199
BTRIEVE/N-multiuser	MS	$465
Corporate PVCS-source control	MS	$359
Flash-Up Windows	PC	$ 79
GSS Graphics Dev't Toolkit	PC	$375
HALO Graphics	PC	$209
Dev't Package	MS	$395
Informix - by RDS	PC	$639
Informix 4GL-application builder	PC	$789
Informix SQL - ANSI standard	PC	$639
Opt Tech Sort - sort, merge	MS	$115
PANEL -	MS	$215
Pfinish - by Phoenix	MS	$229
PolyLibrarian by Polytron	MS	$ 79
PolyBoost - speed I/O, keyboard	PC	$ 69
QMake by Quilt Co.	MS	$ 85
Report Option	MS	$119
Screen Sculptor	PC	$ 95
SRMS - source control	MS	$109
Synergy - create user interfaces	MS	$375
Xtrieve - organize database	MS	$199
ZAP Communications - VT 100	PC	$ 89

Pascal and Supporting

ALICE - learn Pascal	PC	$ 59
Exec - Chain Programs	MS	$ 79
MetaWINDOWS-graphics toolkit bit-mapped, fast	PC	$115
MetaWINDOWS PLUS	PC	$185
Microsoft PASCAL - faster	MS	$189
Pascal Extender	MAC	$ 65
Pascal Pac with Tidy - formatter, utilities	PC	$ 69
Pascal Tools PLUS	PC	$125
Pascal 2 - by Oregon Software, tight, fast	MS	$329
TurboHALO - 150 routines	PC	$ 99

FEATURE

F2C by Solution Systems - Fortran 66 or 77 to C. No max program size $3000. Pioneer. 1000 line max: $ 795

RECENT DISCOVERY

Star Sapphire by Sapien Software. Common LISP compiles/translates to C. Features 8M virtual memory workspace, lexical & dynamic scoping. C Library source, graphics. Flavors, unlimited multidimensional arrays. MS $459

Other Languages

APL*PLUS/PC	PC	$ 429
CCS Mumps - Singleuser	PC	$ 50
CCS Mumps - Multiuser	PC	$ 369
MasterForth - Forth '83	MAC or PC	$ 109
Microsoft MASM	MS	$ 98
Modula-2 - by Pecan	MS	$ 79
Modula-2/86 by Logitech	PC	$ 62
Pasm - by Phoenix	MS	$ 115
PC Forth + - by Lab Micro	PC	$ 199
SNOBOL4 + - great for strings	MS	$ 80
UR/Forth	MS	$ 279

Xenix/Unix

Basic - by Microsoft	$ 239
C-Terp by Gimpel Software	$ 379
Cobol - by Microsoft	$ 639
Cobol Tools - by Microsoft	$ 319
Fortran or Pascal - by Microsoft	$ 439
MicroFocus Lev. II Compact COBOL	$ 795
Panel	$ 539
RM/Cobol	Call
RM/Fortran	Call
Xenix Complete System	$1049

Other Products

386 Assembler/Linker	PC	$ 389
ASMLIB - 170 + routines	PC	$ 129
asmTREE - B + tree file mgmt.	PC	$ 339
Compact Source Print	PC	$ 59
Dan Bricklin's Demo Program	PC	$ 59
Help/Control - on line help	PC	$ 99
Interactive Easyflow-HavenTree	PC	$ 129
Link & Locate - tools to work with Intel and Tektronix projects.	MS	$ 329
LMK - like UNIX make	MS	$ 139
Microsoft Windows	PC	$ 69
Software Development Kit	PC	$ 329
MKS Toolkit - Unix, vi, awk	PC	$ 119
Norton Commander	PC	$ 55
Numerical Analyst by Magus	PC	$ 269
PDisk - cache, tree	PC	$ 125
PLink - 86 PLUS - overlays	MS	$ 319
PMaker - by Phoenix	PC	$ 79
Polymake by Polytron	MS	$ 129
PolyShell by Polytron	MS	$ 119
PolyXREF by Polytron	PC	$ 99
Sapiens V8 - 8M virtual mgr.	PC	$ 300
Taskview - by Sunny Hill Software, ten tasks	PC	$ 55
Tom Rettig's Library - dBASE	PC	$ 89
Tree Diagrammer	PC	$ 59
Visible Computer: 8088	PC	$ 65

Note: Mention this ad. Some prices are specials. Ask about COD and POs. Formats: 3" laptop now available, plus 200 others. UPS surface shipping add $3/item. All prices subject to change without notice.

Call for a catalog, literature, advice and service you can trust

VISA HOURS MasterCard

8:30 AM - 8:00 PM EST.

800-421-8006

THE PROGRAMMER'S SHOP™

Your complete source for software, services and answers

5-D Pond Park Road, Hingham, MA 02043

Mass: 800-442-8070 or 617-740-2510 3/87

"I would like to mention that I appreciate the way that The Programmer's Shop does business. It is indeed refreshing to be able to call and get answers that you can trust in, to questions on various products."

Donald E. Winters
MIS Software Development Inc.

CIRCLE 141 ON READER SERVICE CARD

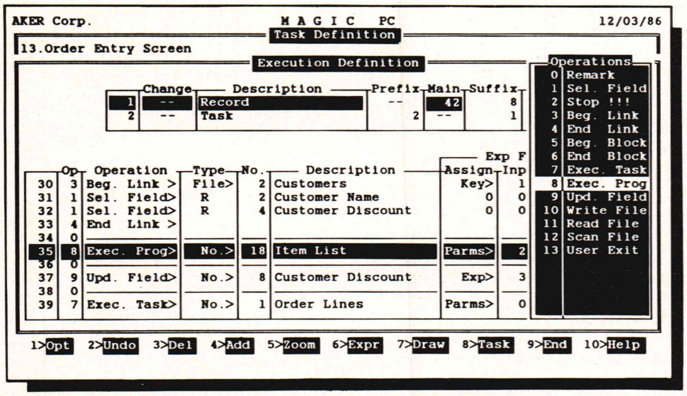
Listing Twenty-six *(Listing continued)*

```
        setnum( &OFFSET, num, offset);
}

/*------------------------------------------------------------*/

pt()
{
        pr_traps();       /* Print all the traps */
}

/*------------------------------------------------------------*/

rd( lstr )
char    *lstr;
{
        /* .rd prompt
         *
         *   Read insertion from standard input until two
         *   newlines in a row are encountered. .rd creates
         *   a macro called "  ". It fills the macro from
         *   standard input, expands the macro, and then
         *   deletes the macro. lstr is a prompt which will
         *   be printed to stderr. A BEL is output as a
         *   prompt whether or not lstr is specified.
         */

        FILE    *oifile;

        fprintf(stderr, "\007");
        if( *lstr )
                fprintf(stderr, "\n%s", lstr);

        oifile = Ifile;
        Ifile  = stdin;
        mcreate( "  " , "\n" );
        Ifile = oifile;

        expand_macro( "  " );
        munlink( "  " );
}

/*------------------------------------------------------------*/

rm(lstr)
char    *lstr;
{
        /* .rm name        Remove the named macro or string.
         *                 if the macro is on the disk, the
         *                 fill will be deleted.
         */

        if( *lstr )
                munlink( lstr );
        else
                err("Missing macro name\n");
}

/*------------------------------------------------------------*/

rr(lstr, rstr)
char    *lstr, *rstr;
{
        /* .rr xx          Remove number register xx. Non-
         *                 existant number registers evaluate
         *         to zero when used in an expression.
         */

        register int    i;

        if( *lstr )
                rm_nreg(lstr) ;
        else
                err("Missing number register name\n" );
}

/*------------------------------------------------------------*/

rs( str )
char    *str;
{
        /* .rs  --  Restore line spacing turned off with
         *          a previous .ns command. Note that
         *          .bp N (the N is required) also works,
         *          as does printing some text.
         */

        Nospace = 0 ;
}

/*------------------------------------------------------------*/

so( lstr )
char    *lstr;
{
        /* .so file  -- get (source) input from named
         *              file. The position in the current
         *   file is remembered and processing will
         *   continue when the sourced file is exhausted.
         */

        register FILE   *fd;

        if( ! (fd = fopen(lstr, "r")) )
                err("Can't open %s\n", lstr );
        else
        {
                process( fd, lstr, 0, 0 );
```

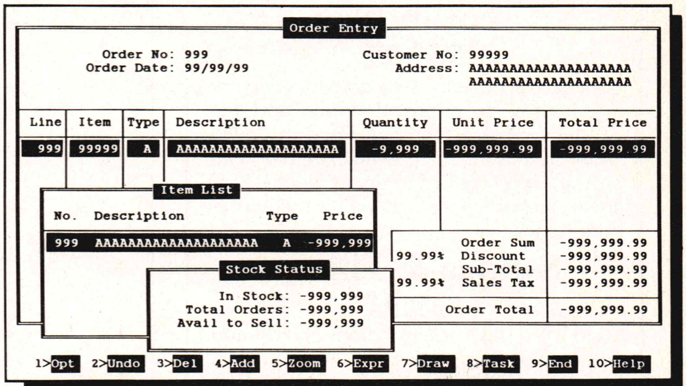
(continued on page 92)

Announcing Magic PC – the first breakthrough for database applications developers in over 20 years:

Now you can develop professional applications 1000% faster than your 4GL or DBMS, totally free from programming, commands and syntax!



A Magic PC program looks as simple as this. To design an application you quickly fill-in menu-driven decision tables **without having to write a single line of code.** For example, just by highlighting the Execute Program operation on this screen and also highlighting the Item List program in the Program Menu, you tell Magic PC to pop-up the Item List window shown in the adjacent screen, when the end-user hits the Zoom key.



Magic PC gives your end-user the power to harness and retrieve data instantly, **without any commands or syntax** because at runtime you already have built-in options to Add, Delete, Modify, Query and get on-the-spot ad-hoc information simply by highlighting selections from menus. Dats validation, security and error-checking are done automatically for you by Magic PC without programming.

Who needs another DBMS?

At last, Magic PC gives you the ultimate applications design tool, far ahead of 4GL's, DBMS and Application Generators.

Magic PC breaks through the language barrier with the revolutionary Un-Language concept:

NO PROGRAMMING, COMMANDS OR SYNTAX!

Free yourself from your programming language

Magic PC makes you, the professional, completely free from the drudgery of procedural programming. No more cryptic commands, syntax or unforgiving procedural structures, because Magic PC does all the programming automatically. There's your competitive edge. The rest is up to you . . .

The Professional Choice

Already an international success, Magic PC is a profit maker and career booster for DP Consultants, System Integrators, VARs, MIS professionals, System Analysts, Programmer Analysts and Software Engineers. If you design PC applications professionally, you can't afford not to Un-Language now.

IBM France: "IBM encourages this introduction and can not help but salute such evolution . . ."

Israeli Air Force: "We were convinced that it was not possible to have a design tool powerful enough to implement real-life applications without a programming language. Magic PC changed our mind . . ."

Jeff Duntemann, PC Tech Journal: "It's probably the best integrated database applications and screen generator that I have ever seen . . .very smooth system, and smoothness comes at a premium these days . . ."

The Magic PC Secret

You're so much more productive with Magic PC because there is **absolutely no programming** to slow you down. You design a Magic PC application by simply filling-in the **Data Dictionary Tables** (Files, Fields, Keys) and the **Task Description Tables** (Operations and Expressions).

Only 13 design **Operations** harness the power of Magic PC. Operations are specific enough to eliminate the need for tiresome syntax, yet elastic enough to produce robust custom applications. Use the Operations to describe **what** you want and Magic PC makes it happen. It's that simple.

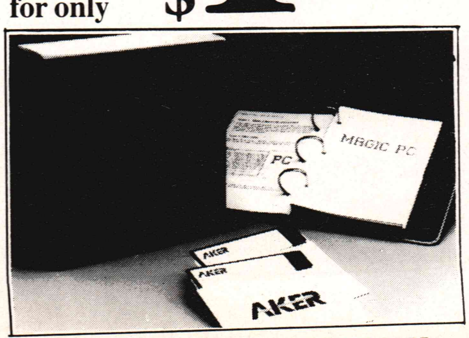
Make Task nesting power available with a single **Execute Task** Operation. This powerful instruction triggers Magic PC to execute and display additional tasks or even external applications through **Window Zooms.** The 3-dimensional effect of Window Zooming lets you probe deep into your application through nested windows and manipulate the data underneath.

You describe a Magic PC Task or Program (composite Tasks) by filling your system analysis flow into the Task Description Tables. Choose the participating Data View, and Magic PC executes your desired Operations. You interface with the Tables by highlighting your selections from pop-up menu-driven windows. There's nothing to edit except your headings.

You're not confined to any particular design sequence as you are with most procedural languages. You can enter and change any Table spontaneously, on the fly, as ideas come to mind and Magic PC automatically maintains the application integrity.

A **Magic Inference Engine** automatically orchestrates your Task Description Tables into a single file of internal **Knowledge Base Rules** for optimum, bug-free performance. Knowledge Base Rules are executed by the **Magic Run** engine for stand-alone runtime operation, or by the **Magic Lan** engine for unrestricted Novell network sharing. You're free to design the Knowledge Base without worrying about the internal structure.

Discover fast, language-free programming at no risk for only $19.95



CIRCLE 369 ON READER SERVICE CARD

See for yourself how fast you can program language-free applications with our low-cost limited offer.

You'll get the full Magic PC software unprotected and limited to 100 records and 450 page documentation complete with a **free** Order Entry sample application. You'll also get our **free** telephone support for 90 days!

And your $19.95 will be credited towards the full $695 Magic PC purchase price. Even if you don't buy Magic PC right away, keep your $19.95 Magic PC Trial as your application prototyping tool at this bargain price.

Our No-Risk Guarantee!

You have our no-risk 30-day money-back guarantee: if you're not completely satisfied for any reason, even Magic PC Trial for $19.95, send it back for a refund.

Order now while supply lasts

Call this toll free number now with your Visa, MasterCard or American Express for immediate delivery, or send the Order Coupon below today to Aker.

1-800-345-MAGIC
in CA call 714-250-1718

VISA MasterCard AMERICAN EXPRESS

Yes, please rush me:

☐ Magic PC Trial	$ 19.95
☐ Magic PC	$695.00
Add shipping	$ 5.00
In CA 6% tax	$_____
Prices valid in US only. Total	$_____

Ship to: ____________________
Address: ____________________
City/ST/Zip: ____________________
Phone: ____________________

AKER

Aker Corp. 18007 Skypark Circle B2, Irvine, CA 92714
(714) 250-1718, Elec. Mail Dialcom 41:AKR 001 Telex 4931184
AKR UI OEM and VAR inquiries are welcome.

Min. requirements PC DOS 2.0, IBM PC or 100% compatible with 512K and hard disk.

C CHEST

Listing Twenty-six *(Listing continued)*

```
                fclose( fd );
        }
}

/*-------------------------------------------------------*/

sp( num, str, offset, dobreak)
char    *str;
{
    /* .sp [N] --   Space down N lines. Default N is 1
    */

    if( dobreak )
        brk();

    if( num > 0 )
        prblank( num );

    else if (num < 0 )
        go_up( num );
}

/*-------------------------------------------------------*/

ss( num )
{
        /* .ss N -- Change the width of a space in the
         *          currently active font to N. Default
         *          N is 1.
         */

        Fonts[ CURFONT ].widths[ ' ' ] = num;
}

/*-------------------------------------------------------*/

ta( str )
char    *str;
{
        /* .ta [A,B,...Z] --  If no argument, clear all tab
         *   [+]B[RCL]        stops. The argument is a list
         *  of tabstops. Each argument can be a specific
         *  column (eg. 9), or an offset from the previous
         *  number (eg. +8), in addition, each number can
         *  be followed by a tab type:
         *      R - Right justified in field
         *      C - Centered in field
         *      L - Left justified in field
         */

        if( *str )
                tabset( str );
        else
                tabclr();
}

/*-------------------------------------------------------*/

tc(str)
char    *str;
{
        /* .tc C    --    MODIFIED NROFF COMMAND --
         *                Set tab expansion character to C.
         * if no C, tab expansion is disabled. This lets
         * us get a ^I or ^A through to the printer control
         * tp.
         */

        if( !*str )
            Tabs_enabled = 0;
        else
        {
            Tabs_enabled = 1;
            Tab = *str ;
        }
}

/*-------------------------------------------------------*/

ti( num, str, off, dobreak )
char *str;
{
        /* .ti N    --    Temporary indent is set to N spaces.
         *                A temporary indent applies only to
         *                the next line of output.
         */

        if( dobreak )
                brk();

        Tempin = num ;
}

/*-------------------------------------------------------*/

tl( str )
char    *str;
{
        /* .tl /A/B/C/  Print a 3-part title with A left
         *              justified, B centered, and C right
         *  justified. The / can be any character. The
         *  title is printed at the current page offset
         *  but indent is ignored, the length is defined
         *  with the .lt command.
         */

        title( str );
}
```

(continued on page 94)

THE PROGRAMMER'S SHOP

provides complete information, advice, guarantees and every product for Microcomputer Programming.

AUTO-INTELLIGENCE

The First Automatic Knowledge Acquisition System

List: $990 Our: $749

IntelligenceWare, Inc.

9800 S. Sepulveda Blvd. Suite 730
Los Angeles, CA 90045
Telephone: [213] 417-8896; Fax [213] 417-8897

CIRCLE 301 ON READER SERVICE CARD

WANT TO ADD

WINDOWS, ICONS, FONTS, FAST GRAPHICS, DIALOG BOXES, PROCESS MANAGEMENT, AND DEVICE INDEPENDENCE

TO YOUR IBM PC PROGRAMS?

If you have ever wished that you could develop stunning Macintosh-like programs on the IBM PC without the overhead of an enormous operating environment like Windows or GEM, then you need the **SYNERGY DEVELOPMENT TOOLKIT, from Matrix Software.**

Using a memory resident runtime module only **20K** in size (versus as much as 300K for Windows), you can develop applications with: overlapped and tiled windows; pull-down menus with half intensity options and automatic sizing; fast graphics function calls to draw shapes, lines, boxes, and create intricate fill patterns in both regular and irregular areas; have full device independence (drivers for numerous devices, including CGA, EGA and Hercules are included); sophisticated text input and output, with fonts in different styles and sizes; full keyboard support (your programs won't need a mouse!) and powerful mouse support; and process management calls to efficiently manipulate system resources.

The Toolkit has gateways to support the following languages:

- Turbo Pascal
- IBM/MS Pascal
- Microsoft & Lattice C
- Macro Assembler
- Basic
- dBASE II/III Compilers

In addition, the Toolkit includes a powerful collection of tools including a graphics resource editor for developing your own icons and fonts.

NEW! The Toolkit also **includes a free copy of Synergy Layout**, a revolutionary software development tool that dramatically increases your productivity by actually generating bug-free source code in Macro, C, and Turbo Pascal.

For further information, contact Matrix Software at [617] 567-0037.

List: $395
Our: $349

CIRCLE 302 ON READER SERVICE CARD

MUMPS

Now quickly and easily develop multi-tasking applications on your PC!

Now get the speed, power and flexibility of MUMPS in a truly concurrent processing environment with COMP Computing Standard MUMPS (CCSM).

CCSM provides the programmer with these benefits:

* Save time and money by writing Applications in 1/3 to 1/5 the amount of code.
* Fast data access with B-Tree File Structure and Automatic disk caching.
* Unlimited program size and variable space with advanced virtual memory system.
* 100% portable from micros to minis to mainframes.
* Easily write multi-tasking and multiuser applications.

And with MUMPS, all you need to do to start a background process is: "JOB ^ROUTINE". No memory allocation, no table set-up, no hassles; the system handles <u>everything</u>.

CCSM is also available in a multi-tasking version for the Macintosh.

Multi-Tasking:	List - $149.95	Ours - $129
Multiuser:	List - $450.00	Ours - $369

For a demonstration diskette of CCSM, send two dollars to the address below.

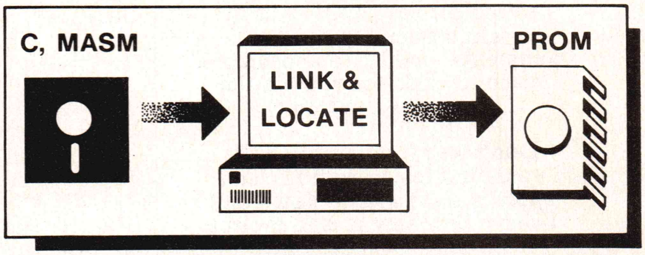
MGlobal
1601 Westheimer, Suite 201
Houston, Texas 77006

1-800-257-8052

In Texas: 1-713-529-2576

CIRCLE 303 ON READER SERVICE CARD

FIRMWARE DEVELOPMENT



LINK & LOCATE enables PC users to produce ROM-based firmware for 8086/87/186 from object files generated by popular C compilers, such as from Wizard, Microsoft and Lattice, and MASM assembler from Microsoft. Provides full control of segment placement anywhere in memory. Supports output of Intel HEX file for PROM programmers, Intel OMF absolute object file for symbolic debuggers and in-circuit emulators. Includes Intel compatible linker, locator, librarian, hex formatters and cross reference generator. $350.

SYSTEMS & SOFTWARE INC.

Our: $329

3303 Harbor Blvd., C11, Costa Mesa, CA 92626
Phone (714) 241-8650 FAX (714) 241-0377 TWX 910-695-0125

CIRCLE 304 ON READER SERVICE CARD

Call Today for FREE detailed information or try Risk-Free for 31 days.

800-421-8006

HOURS: 8:30 A.M. - 8:00 P.M. E.S.T.

THE PROGRAMMER'S SHOP™
Your complete source for software, services and answers
5-D Pond Park Road, Hingham, MA 02043
Mass: 800-442-8070 or 617-740-2510 2/87

Listing Twenty-six *(Listing continued)*

```
/*------------------------------------------------------*/

tm(lstr )
char    *lstr;
{
        /* .tm str   --  Print string to standard error.
         */

        fprintf(stderr, lstr );
}

/*------------------------------------------------------*/

tp()
{
        /* .tp -- NOT AN NROFF COMMAND --
         *       prints the current tab stops the current
         *       line length
         */

        tabprint();
}

/*------------------------------------------------------*/

ul( num, str, offset)
char    *str;
int     num, offset;
{
        /* .ul [N]        -- Underline the next N input lines,
         *                   if N is absent. Only alphanumeric
         *                   characters are underlined, punction,
         *                   spaces, etc., will not be.
         */

        setnum( &Num_under, num, offset );
}

/*------------------------------------------------------*/

vd( num, str, offset, dobreak, tail )
char    *str;
char    *tail;
{
        /* .vd <up str> N <down str> -- NOT NROFF COMMAND --
         *
         * defines strings to send printer cursor up or
         * down by 1/N lines.
         */

        static char dnstr[81];
        static char upstr[81];

        strncpy( Up_str = upstr, str , 80 );
        strncpy( Dn_str = dnstr, tail, 80 );

        Vs_amt = num   ;
}

/*------------------------------------------------------*/

wa( num )
{
    /* .wa [N]  -- NOT AN NROFF COMMAND --
     *
     *  Waits for about N seconds (at most N + 1).
     *
     *  If N == 0 or if no argument a prompt is
     *  printed and the program waits for you to
     *  type Enter.
     */

    int sec, osec, garbage ;

    if( !num )
    {
        fprintf(stderr, "\n\007\nType CR to continue..." );
        fprintf(stderr, "%c\n", getch() );
    }
    else
    {
            num++;

            fprintf( stderr, "\n");

            while( --num >= 0 )
            {
                fprintf( stderr, "waiting: %02d\r", num );

                time(&garbage, &garbage, &osec, &garbage );

                do {
                    time(&garbage, &garbage, &sec, &garbage );

                } while( osec == sec );
            }

            fprintf( stderr, "\n");
    }
}

/*------------------------------------------------------*/

wh( num, str )
char    *str;
{
        /* .wh N X   --  Set output line trap. The macro X
```

```
         *             is executed imediately after
         *  printing line N on the current page. If N==0
         *  the the trap is sprung at the top of page
         *  (above line 1). If X is absent, the trap at
         *  line N is removed. If N is negative then the
         *  trap is set relative to the bottom of the
         *  page length (as set with .pl) that was in
         *  .wh was executed. The macro will replace any
         *  previously installed macro, macros do not
         *  shadow one another as in the real nroff.
         */

        set_linetrap(str, num);
}

/*-------------------------------------------------------*/

ws( num )
{
        /*  .ws N        -- NOT AN NROFF COMMAND --
         *
         *  Enables wordstar-mode output:
         *
         *  N == 0 (or missing) wordstar mode disabled
         *  N == 1 all single newlines mapped to to wordstar
         *          soft carriage returns. \n\n is printed as
         *          two hard carriage returns, however.
         *  N == 2 like N==1 except that single carriage
         *          returns are replaced with space characters.
         *
         *  Note that you'll also want to do the following:
         *
         *  .po 0                  \"  No page offset
         *  .bd \x02 \x02          \" ^B for bold
         *  .ud \x13 \x13          \" ^S for underline
         *  .od \x18 \x18          \" ^X for overstrike
         */

        Wordstar = num;
}

/*-------------------------------------------------------------
 *  List of legal commands. Command names must be listed in
 *  alphebetical (ASCII) order. The subroutine pointed to by
 *  the "rout" field is called when the command is recognized.
 *  Command format types (column labeled "tp" below) are:
 *
 *  Type 0:   .xx [<str>]          default used for <str>
 *  Type 1:   .xx N [<str>]        default used for N
 *  Type 2:   .xx <str> N [<tail>] default used for N
 *  Type 3:   .xx <lstr> <rstr>    default used for <lstr>
 *
 * If the default field is empty ("") it is passed to the
 * subroutine as an empty string in type 0 and 1 commands
 * and as a 0 in type 1 and 2 commands.
 *
 * If the "inhib" field is set then the command won't be
 * executed when input has been inhibited by if/else
 * processing. Only control-flow commands are uninhibited.
 * To change a command name you need only modify the table.
 * The exception is .{ because this command is generated
 * explicitly when a \{ is encountered.
 *
 * Consult command() in nrcmd.c for more details on how the
 * table is used.                                     (}})
 */

CTAB    Cmdtab[] =
{
   /*     name              rout.,  tp, inhib, default      */

        { "ad"   ,          ad      , 0, 1  , ""             },
        { "af"   ,          af      , 3, 1  , ""             },
        { "am"   ,          am      , 3, 1  , ""             },
        { "as"   ,          as      , 3, 1  , ""             },
        { "bd"   ,          bd      , 3, 1  , ""             },
        { "bo"   ,          bo      , 1, 1  , "1"            },
        { "bp"   ,          bp      , 1, 1  , ""             },
        { "br"   ,          br      , 0, 1  , ""             },
        { "c2"   ,          c2      , 0, 1  , "`"            },
        { "cc"   ,          cc      , 0, 1  , "."            },
        { "ce"   ,          ce      , 1, 1  , "1"            },
        { "cf"   ,          cf      , 0, 1  , ""             },
        { "ch"   ,          ch      , 2, 1  , "0"            },
        { "cm"   ,          cm      , 0, 1  , ""             },
        { "cu"   ,          cu      , 1, 1  , "1"            },
        { "da"   ,          da      , 0, 1  , ""             },
        { "db"   ,          db      , 0, 1  , ""             },
        { "de"   ,          de      , 3, 1  , ""             },
        { "df"   ,          df      , 3, 1  , ""             },
        { "di"   ,          di      , 0, 1  , ""             },
        { "ds"   ,          ds      , 3, 1  , ""             },
        { "dt"   ,          dt      , 1, 1  , "0"            };
        { "ec"   ,          ec      , 0, 1  , "\\"           },
        { "el"   ,          el      , 0, 0  , ""             },
        { "em"   ,          em      , 3, 1  , ""             },
        { "eo"   ,          eo      , 0, 1  , ""             },
        { "ev"   ,          ev      , 0, 1  , ""             },
        { "ex"   ,          ex      , 0, 1  , ""             },
        { "fi"   ,          fi      , 0, 1  , ""             },
        { "ft"   ,          ft      , 0, 1  , "R"            },
        { "hd"   ,          hd      , 2, 1  , ""             },
        { "hy"   ,          hy      , 1, 1  , "1"            },
        { "id"   ,          id      , 3, 1  , ""             },
        { "ie"   ,          ie      , 3, 0  , ""             },
        { "if"   ,          iff     , 3, 0  , ""             },
        { "ig"   ,          ig      , 0, 1  , ""             },
        { "in"   ,          in      , 1, 1  , "0"            },
```

(continued on page 97)

Listing Twenty-six *(Listing continued)*

```
        { "it"  ,       it      , 1, 1  , ""        },
        { "lc"  ,       lc      , 1, 1  , "."       },
        { "ll"  ,       ll      , 1, 1  , "80"      },
        { "ls"  ,       ls      , 1, 1  , "1"       },
        { "lt"  ,       lt      , 1, 1  , "80"      },
        { "mc"  ,       mc      , 2, 1  , ""        },
        { "mf"  ,       mf      , 3, 1  , ""        },
        { "ml"  ,       ml      , 0, 1  , ""        },
        { "na"  ,       na      , 0, 1  , ""        },
        { "nb"  ,       nb      , 0, 1  , ""        },
        { "ne"  ,       ne      , 1, 1  , "1"       },
        { "nf"  ,       nf      , 0, 1  , ""        },
        { "nh"  ,       nh      , 0, 1  , ""        },
        { "nm"  ,       nm      , 0, 1  , ""        },
        { "nr"  ,       nr      , 2, 1  , "0"       },
        { "ns"  ,       ns      , 0, 1  , ""        },
        { "od"  ,       od      , 3, 1  , ""        },
        { "os"  ,       os      , 1, 1  , "1"       },
        { "ou"  ,       ou      , 0, 1  , ""        },
        { "pc"  ,       pc      , 0, 1  , "%"       },
        { "pl"  ,       pl      , 1, 1  , "66"      },
        { "po"  ,       po      , 1, 1  , "0"       },
        { "pt"  ,       pt      , 0, 1  , ""        },
        { "rd"  ,       rd      , 0, 1  , "\007"    },
        { "rm"  ,       rm      , 0, 1  , ""        },
        { "rr"  ,       rr      , 0, 1  , ""        },
        { "rs"  ,       rs      , 0, 1  , ""        },
        { "so"  ,       so      , 0, 1  , ""        },
        { "sp"  ,       sp      , 1, 1  , "1"       },
        { "ss"  ,       ss      , 1, 1  , "1"       },
        { "ta"  ,       ta      , 0, 1  , ""        },
        { "tc"  ,       tc      , 0, 1  , " "       },
        { "ti"  ,       ti      , 1, 1  , "0"       },
        { "tl"  ,       tl      , 0, 1  , ""        },
        { "tm"  ,       tm      , 0, 1  , "\007"    },
        { "tp"  ,       tp      , 0, 1  , ""        },
        { "ul"  ,       ul      , 1, 1  , "1"       },
        { "vd"  ,       vd      , 2, 1  , ""        },
        { "wa"  ,       wa      , 1, 1  , ""        },
        { "wh"  ,       wh      , 1, 1  , ""        },
        { "ws"  ,       ws      , 1, 1  , ""        },
        { "{"   ,       sblock  , 0, 0  , ""        },
        { "}"   ,       eblock  , 0, 0  , ""        }
};

int     Ctabsize = (sizeof(Cmdtab) / sizeof(*Cmdtab));
```

End Listing Twenty-six

Listing One

```
/*-----------------------------------------------------------
 *      STAT.C          Statistics routines:
 *
 *  newsample( n )  Add a new sample to the mean/average
 *                  totals.
 *  running_mean()  Returns the running mean of the samples.
 *  true_mean()     Returns the true mean of the samples.
 *  deviation()     Returns the standard deviation from the
 *                  running mean.
 *  reset_mean(n)   Resets everyting to 0. 'n' is the boxcar
 *                  length that will be used for subsequent
 *                  samples. If n is 0, the default length of
 *                  4 is used.
 */

#define DEF_BOXLEN 4

static unsigned long    Average    = 0;
static unsigned long    Numnums    = 0;
static unsigned long    Mean_total = 0;
static unsigned long    Mean       = 0;
static unsigned long    Dev_total  = 0;
static unsigned long    Dev        = 0;
static unsigned int     Boxlen     = DEF_BOXLEN ;

/*----------------------------------------------------------*/

void    newsample( n )
{
    /*  Add a new point into the various mean and deviation
     *  variables.
     */

    register unsigned long  dif;

    Average += n;
    Numnums++;

    Mean_total -= Mean ;            /* find running mean  */
    Mean_total += n;
    Mean = Mean_total >> Boxlen;

```

(continued on next page)

Listing One *(Listing continued)*

```
    dif        = abs( Mean - n );  /* Distance  to point */
    dif       *= dif;              /* square it.         */

    Dev_total -= Dev;              /* find average       *
    Dev_total += dif;              /* difference         */
    Dev = Dev_total >> Boxlen;
}

/*-------------------------------------------------------*/

int running_mean()      /* Return the current running mean */
{
        return Mean;
}

/*-------------------------------------------------------*/

int true_mean()          /* Return the current true mean   */
{
        return Average / Numnums;
}

/*-------------------------------------------------------*/

int deviation()        /* Return the current standard      */
{                      /* deviation from the running mean. */

        extern double sqrt();
        return (int) sqrt( (double)Dev );
}

/*-------------------------------------------------------*/

void reset_mean( boxcar_val )
{
        /* Reset various global variables to their initial
         * values, "boxcar_val" is used for the boxcar
         * width. It is a shift value, not a true width. If
         * it's 0, the default value of 4 is used instead.
         */

        Average    = 0;
        Numnums    = 0;
        Mean_total = 0;
        Mean       = 0;
        Dev_total  = 0;
        Dev        = 0;
        Boxlen     = boxcar_val ? boxcar_val : DEF_BOXLEN ;
}

/*-------------------------------------------------------*/

#ifdef MAIN

#define NUMSAMPLES      50

test( how )
{
    /*      how = 0        Straight line.
     *      how = 1        Triangle
     *      how = 2        Random
     */

    int     i, j, count, m, a, d, dir = 1 ;

    for( count = 0;  count += dir;  )
    {
        if( count == NUMSAMPLES )
            dir = -1;

        newsample( i = (( how == 2 ) ? (rand() % NUMSAMPLES) :
                        ( how == 1 ) ? (count              ) :
                                       (NUMSAMPLES / 2     )));
        m = running_mean();
        d = deviation();
        a = true_mean();

        for( j = 1; j <= NUMSAMPLES; j++ )
        {
            if( j>i && j>m && j>a )
                break;

            if      ( j == i )      printf("*");
            else if ( j == m )      printf("m");
            else if ( j == a )      printf("a");
            else                    printf(" ");
        }
```

```
        printf("\n");
    }
}

/*------------------------------*/

main()
{
    char    buf[80];
    int     how;

    reset_mean( 4 );
    test( 0 );
    printf("Straight line with length 16 boxcar\n\f");

    reset_mean( 2 );
    test( 1 );
    test( 1 );
    printf("Triangle wave with length 4 boxcar\n\f");

    reset_mean( 4 );
    test( 1 );
    test( 1 );
    printf("Triangle wave with length 16 boxcar\n\f");

    reset_mean( 6 );
    test( 1 );
    test( 1 );
    printf("Triangle wave with length 64 boxcar\n\f");

    reset_mean( 2 );
    test( 2 );
    printf("Random input with length 4 boxcar\n\f");

    reset_mean( 4 );
    test( 2 );
    printf("Random input with length 16 boxcar\n\f");

    reset_mean( 6 );
    test( 2 );
    test( 2 );
    printf("Random input with length 64 boxcar\n\f");

#ifdef NEVER

    while( 1 )
    {
        printf( "triangle, random, or linear (r/t/l)?" );
        gets( buf );

        how = ( *buf == 'r' ) ? 2 : ( *buf == 't' ) ? 1 : 0;

        printf("Boxcar length? ");
        gets( buf );
        test( atoi(buf), how );
    }

#endif
}

#endif
```

End Listing One

(Softstrips are on page 101.)

FILES:
STAT.C

Both of these Softstrips by Cauzin Systems contain complete versions of Listing One for this month's C Chest. The strip on the left is in high-density format, and the one on the right is in medium density.

FILES:
STAT.C

1 | 2 |

| 2 | 3 |

Statistical Applications of Digital Low-Pass Filters, Exec Bug in Microsoft C

In keeping with this issue's dual theme of music and scientific programming, this month's C Chest looks at a set of subroutines that have applications in both worlds.

Two of the more useful statistical functions are the arithmetic mean and standard deviation from that mean. Given some set of data points, the mean is just the average value of the points. The standard deviation is the average distance from the mean to the various data points. To be more precise, it's the square root of the arithmetic mean of the squares of the distance of each data point from the mean. The standard deviation is usually used as a measure of the dispersion of the various points around the mean—a sort of average-error indication—the higher the standard deviation, the larger the average error. In other words, it tells you how close the real data is to the average, or expected, value.

Computing a true mean and standard deviation can be quite difficult, especially if the number of data points is large or if these points have large values. It's easy to run out of precision because you have to sum all the points in the input sample before you can divide. Moreover, you can't compute either the true mean or deviation until you've collected all the data points; you can't do it on the fly. Fortunately, there are ways to approximate the mean and deviation that don't have these limitations. I'll look at one of these here—an exponential smoothing function, or digital low-pass filter.

by Allen Holub

All sounds, regardless of the waveshape, can be broken up into the sum of a series of sine waves. The lowest-frequency component is the fundamental, and the higher-frequency components are called harmonics, or partials. A low-pass filter removes the higher-frequency components from a particular sound and leaves the lower-frequency ones intact. Most stereos have a low-pass filter built into their tone controls. When you turn the treble control all the way down, all the high-frequency components of the sound are removed, leaving only the low-frequency components.

Mathematically, a low-pass filter is a "leaky integrator." It computes the integral, not of a whole curve, but of the most recently seen parts of the curve. You can look at an integral as the area under the curve—the area of a shape bounded on one side by some function and on the other by the x axis. Consequently, you can get a good approximation of the integral from a set of discrete, equally spaced points by summing all the distances from the x axis to the points. That is, if you consider each point to represent a box whose height is the distance to the curve and whose width is 1, the area of each box is the height and the total area is just the sum of the heights. If the distance between the points is not 1, you can compensate by multiplying the sum by the actual distance.

To use the integral as a low-pass filter, you limit the range of the integration, including in the sum only those points in an n-point wide window. The wider the window, the lower the cutoff frequency of the filter. That is, when a larger part of the curve is included in the sum, the parts of the curve that change fastest (the higher-frequency components) tend not to affect the sum as much as the components that change more slowly. Returning to statistics, the value of the arithmetic mean is just the integral divided by the number of points.

For those of you who are electronically inclined, a standard low-pass filter circuit is shown in Figure 1, page 105. If R_{leak} is removed, this circuit is an integrator. That is, the various voltages present in the input will all be summed in the capacitor, C. If the input is positive, the capacitor is charged; if the input is negative, the capacitor is discharged. The integrator has been modified, however, by putting a leakage resistor (R_{leak}) around the capacitor. This resistor causes the capacitor to discharge slowly, even when there's no input. The value of the resistor determines the rate of discharge, which in turn determines the width of the window.

Digital low-pass filters can be implemented in several ways. The most straightforward is actually to average some set of contiguous input points. Every time you acquire a new input point, you discard the oldest point, add the new one, and refigure the average. The more points that are included in the average, the lower the cutoff frequency. A fixed-length circular queue is a good choice of data structures. You can treat it as a queue when you insert new points and as an array when you average the points. This method, called a boxcar average, isn't too useful in practice, however. It's just too hard to keep all those points around, add them up, and then divide by the number of points every time you get a new input sample.

A more practical method that is similar to a true boxcar average is exponential smoothing. It's best explained with an example. Say you have a length-16 boxcar. Every time you get a new input sample, you subtract 1/16th of the current value from the average and then add

C Programmers!

db_VISTA™: high-speed Database written exclusively for C NOW offers SQL-based Query

"db_VISTA™ has proved to be an all-round high performer in terms of fast execution. . ."

John Adelus, Hewlett-Packard Ltd./Office Productivity Division

High-speed data retrieval and access... just two benefits of using RAIMA's network model DBMS, db_VISTA. Combine these benefits with those of C—speed, portability, efficiency, and you begin to understand db_VISTA's real measure... performance.

db_QUERY™: new simplicity retains performance!

db_QUERY, our new C-linkable, SQL-based, ad-hoc query and report writing facility ... provides a simple, relational view of db_VISTA's complex network database. No longer will you give up performance for simplicity ... combine db_QUERY with db_VISTA ... you have both!

Independent Benchmark proves High-Speed model 2.76 times faster

An independent developer benchmarked db_VISTA against a leading competitor. Eleven key retrieval tests were executed with sequentially and randomly created key files.

*Result of 11 Key Retrieval Tests

db_VISTA	:671.24 seconds
Leading Competitor	:1,856.43 seconds

db_VISTA's high-speed network database model lets you precisely define relationships to minimize redundant data. Only those functions necessary for operation are incorporated into the run-time program.

Application Portability Complete Source Code

For maximum application portability, every line of db_VISTA's code is written in C and complete source code is available. db_VISTA operates on most popular computers and operating systems. So whether you write applications for micros, minis, or mainframes...db_VISTA is for you.

How db_VISTA works...

Design your database and compile your schema file with the database definition language processor. Develop application programs, making calls to db_VISTA's C functions. Edit and review your database using the Interactive Database Access utility. Compile and link your C program with the db_VISTA run-time library, and your application is ready to run.

Multi-user and LAN capability

Information often needs to be shared. db_VISTA has multi-user capability and supports simultaneous users in either multi-tasking or local area networking environments, allowing the same C applications to run under UNIX, MS-DOS, and VAX VMS.

Royalty-Free Run-Time

Whether you're developing applications for a few customers, or for thousands, the price of db_VISTA or db_QUERY is the same. If you are currently paying royalties for a competitor's database, consider switching to db_VISTA and say goodbye to royalties.

FREE Technical Support For 60 days

Raima's software includes free telephone support and software updates for 60 days. Technical support personnel are available to answer questions about our software or yours.

30-Day Money-Back Guarantee

Try db_VISTA for 30 days and if not fully satisfied, return it for a full refund.

Price Schedule

	db_VISTA	db_QUERY
☐ Single-user	$ 195	$ 195
☐ Single-user w/Source	$ 495	$ 495
☐ Multi-user	$ 495	$ 495
☐ Multi-user w/Source	$ 990	$ 990
NEW:		
☐ VAX Multi-user	$ 990	$ 990
☐ VAX Multi-user w/Source	$1980	$1980

Call Toll-Free Today!
1 (800) db-RAIMA
(that's 1-800-327-2462)
---OR Call 1-206-828-4636

Read what others say...

"If you are looking for a sophisticated C programmer's database, db_VISTA is it. It lets you easily build complex databases with many interconnected record types. Raima's customer support and documentation is excellent. Source code availability and a royalty-free run-time is a big plus."
Dave Schmitt, President Lattice, Inc.

"My team has developed a sophisticated PC-based electronic mail application for resale to HP customers. db_VISTA has proved to be an all-round high performer in terms of fast execution, flexibility and portability, and has undoubtedly saved us much time and development effort."
John Adelus, Hewlett-Packard Ltd. Office Productivity Division

"On the whole, I have found db_VISTA easy to use, very fast with a key find, and powerful enough for any DBMS use I can imagine on a microcomputer."
Michael Wilson, Computer Language

db_VISTA Version 2.2

Features

- **Multi-user** support allows flexibility to run on local area networks
- **File structure** is based on the B-tree indexing method and the network database model
- **Run-time size**, variable—will run in as little as 64K, recommended RAM size is 256K
- **Transaction processing** assures multi-user database consistency
- **File locking** support provides read and write locks on shared databases
- **SQL-based** db_QUERY is linkable
- **File transfer** utilities included for ASCII, dBASE optional
- **Royalty-free** run-time distribution.
- **Source code** available.

Database Record and File Sizes

- Maximum record length limited only by accessible RAM
- Maximum records per file is 16,777,215
- No limit on number of records or set types
- Maximum file size limited only by available disk storage
- Maximum of 255 index and data files

Keys and Sets

- Key length maximum 246 bytes
- No limit on maximum number of key fields per record—any or all fields may be keys with the option of making each key unique or duplicate
- No limit on maximum number of fields per record, sets per database, or sort fields per set
- No limit on maximum number of member record types per set

Operating System & Compiler Support

- **Operating systems:** MS-DOS, PC-DOS, UNIX, XENIX, SCO XENIX, UNOS, ULTRIX, VMS
- **C compilers:** Lattice, Microsoft, IBM, DeSmet, Aztec, Computer Innovations, XENIX and UNIX

Utilities

- Database definition language processor
- Interactive database access utility
- Database consistency check utility
- Database initialization utility
- Multi-user file locks clear utility
- Key file build utility
- Data field alignment check utility
- Database dictionary print utility
- Key file dump utility
- ASCII file import and export utility

*The benchmark procedure was adapted from "Benchmarking Database Systems: A Systematic Approach" by Bitton, DeWitt and Turbyfill, December 1983.

DD57

RAIMA™ CORPORATION

High-Speed Programming Tools, Designed for Portability

Call Toll-Free Today!

1 (800) db-RAIMA

(that's 1-800-327-2462)

3055–112th Avenue N.E. • Bellevue, WA 98004 USA • (206) 828-4636 Telex: 6503018237 MCI UW

(continued from page 102)

1/16th of the new sample's value to the average. In pseudocode:

```
Width = 16;
while(1)
{
  Boxcar -= Boxcar   / Width;
  Boxcar += input( ) / Width;
}
```

Because you're always subtracting 1/16th of the current boxcar, the contents change exponentially. (Think about what happens if all the input samples suddenly go to zero; the value of *Boxcar* will decrease by 1/16th of its former value every time through the loop.) Figure 2, below, shows a simple case that demonstrates the exponential aspect of the algorithm going in the other direction. I'm using a length-16 boxcar, and the input set defines a straight, horizontal line (marked with asterisks). The mean (marked with *m*s) starts out at 0 and gradually (exponentially) converges on the input line. If the boxcar had been smaller, it would have converged faster.

You'll notice a few similarities between this algorithm and the electronic equivalent in Figure 1. In particular, *Width* takes the place of R_{leak} in the circuit. That is, changing the width of the window effectively changes the cutoff frequency of the filter. If *Width* is 1, there will be no filtering; the input will just pass through to the output. When *Width* is 16, no change in the input that happens in fewer than 16 samples will make it through to the output. That is, only those changes in the input that happen more slowly than the rate at which the average changes will make it through to the output.

Exponential smoothing can be implemented naively using the above algorithm, but as in most such algorithms, a little thought gives you both more efficiency and more accuracy. There are two changes that are easy to make. First, by limiting the boxcar length to a power of 2, you can replace the divides with right shifts. Second, rather than have the *Boxcar* variable hold the mean itself, you can make it hold 16 times the mean, thereby eliminating one of the divides and giving you better precision at the same time. The modified algorithm is:

```
while(1)
{
  Boxcar -= Mean ;
  Boxcar += input( ) ;
  Mean    = Boxcar >> 4;
}
```

The boxcar is updated with every input sample. The mean is computed by dividing *Boxcar* by 16. (A right shift of 4 bits is a divide by 16.) When you subtract *Mean* from *Boxcar*, you're effectively subtracting 1/16th of the mean from itself. You don't loose any precision here, though, as you would if you divided before subtracting.

Now let's apply the boxcar algorithm in a digital-filter application. Figure 3, page 106, shows the algorithm being used on a triangle wave.

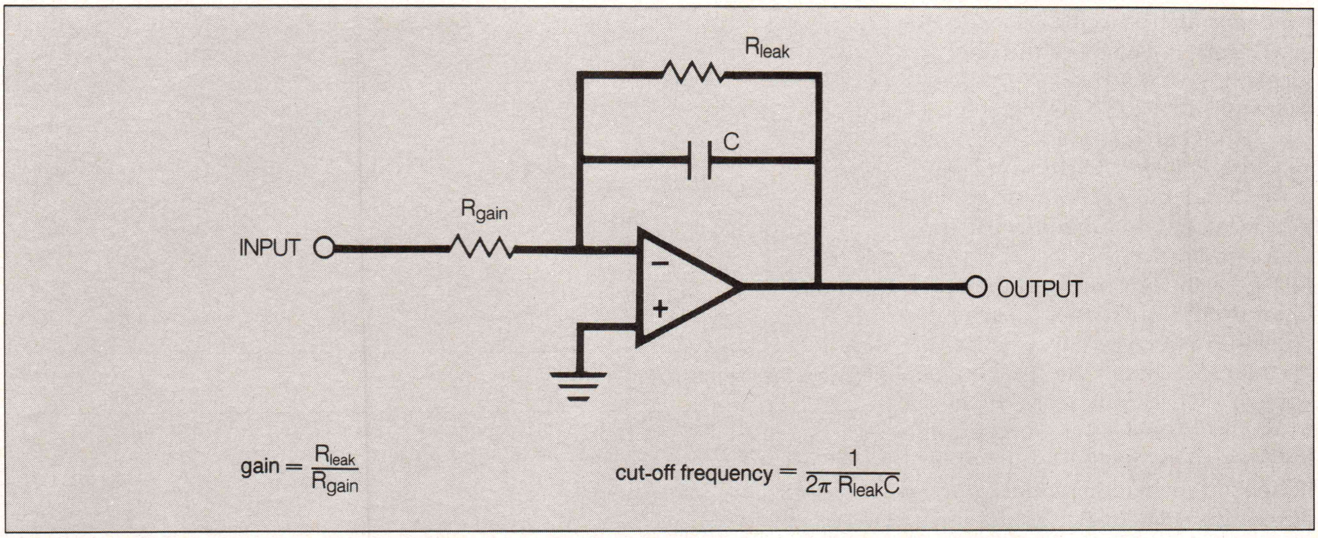


Figure 1: *Analog low-pass filter or leaky integrator*

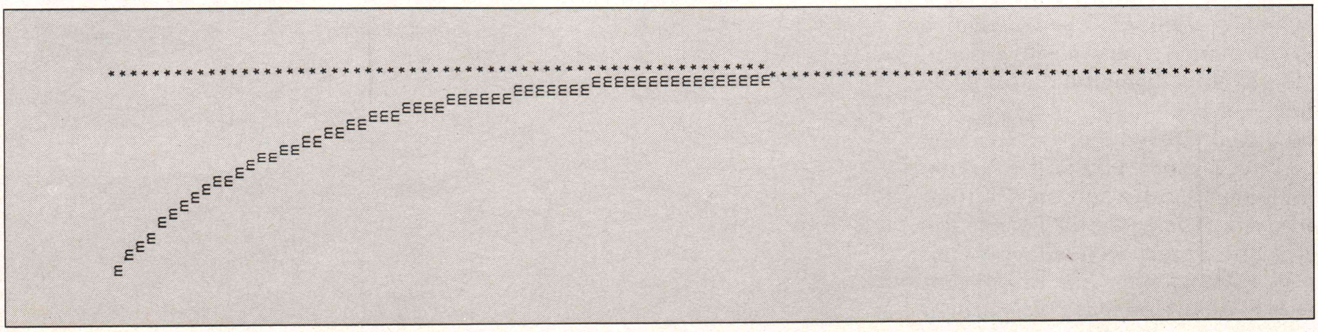


Figure 2: *Behaviour of an exponential smoothing function on a straight line*

(continued from page 105)

The input data is the same in all three parts of the figure, but the length of the boxcar is different. In all three cases, the input is marked with asterisks, the boxcar average with sideways *m*s, and the true average (arithmetic mean) with sideways *a*s. The input data contains 100 points per cycle (200 points in the entire graph). Figure 3a shows a boxcar of length 4. Here there's not much to see. The points on the input triangle are slightly rounded, but that's about it. It's interesting to note the slight phase shift between the input and output. This phase shift is also a characteristic of analog low-pass filters (built with capacitors, resistors, and such). It's caused by the amount of time required to charge the capacitor in the analog filter. In the digital equivalent, it's the amount of time needed for the mean to ramp up to the output value. So, you'd expect the phase shift to increase as the width of the boxcar (or the value of the capacitor) increases.

Figure 3b shows the same input, now being filtered with a length-16 boxcar. Here the effect of the filter is noticeable. Most of the high-frequency harmonics of the triangle have been removed, leaving something very close to a sine wave representing the fundamental. Note that the phase shift has increased with the amount of filtering, just as expected.

In Figure 3c, I'm using a length-64 boxcar. Pretty much all the harmonics now have been removed, and I've just about eliminated the fundamental, too. If I were to increase the boxcar length to 100—the number of points per cycle in the input—then I'd filter out the fundamental entirely, leaving a straight horizontal line. It makes sense if you think about it. If you average a complete cycle of a sine wave, there will be a negative point to match every positive point, so the average over the entire cycle will be zero.

So the cutoff frequency of the low-pass filter is a function of the sample rate (in data points) and the width of the boxcar. The wider the boxcar, the lower the cutoff frequency. When the boxcar has the same number of points in it as a sine wave of a particular frequency, that frequency won't make it through the filter at all. Computing the actual cutoff frequency of a digital filter—one that's not a nice power of 2, for instance—is nontrivial, however. If you're interested, there's a good description in Hal Chamberlin's book (cited in the bibliography), pages 481–495.

Now, let's bring all this back into the realm of statistics. Looking again at Figure 2, because the input data is a horizontal straight line, the arithmetic mean is the same straight line. Consequently, in this example the

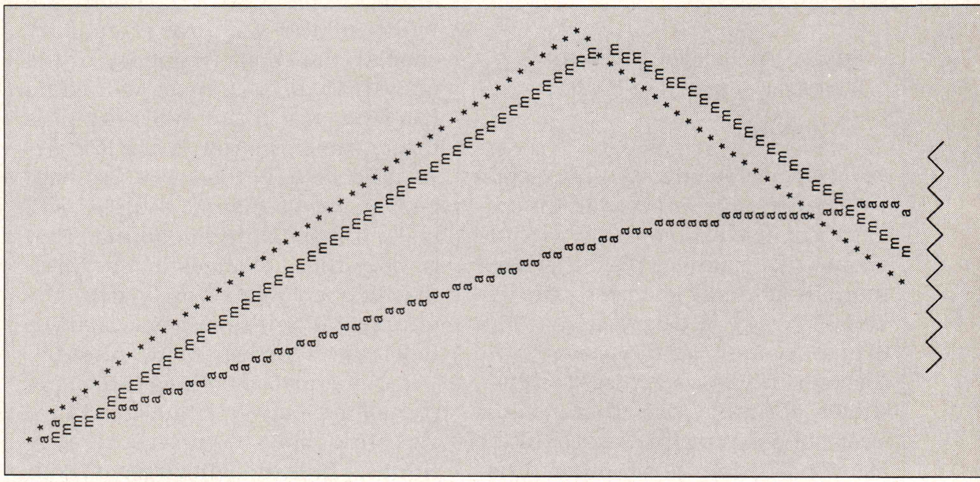


Figure 3: *Varying the window width in an exponential smoothing function.*

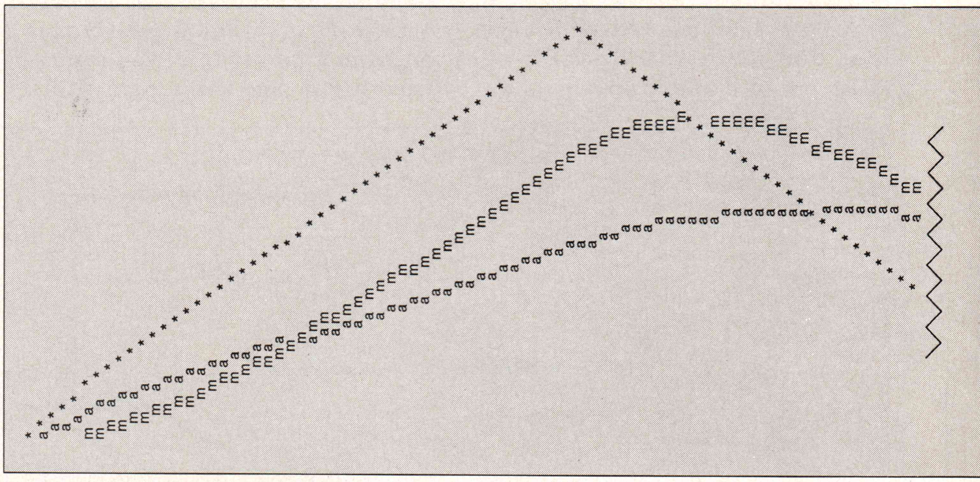


Figure 3b: *Length-16 window*

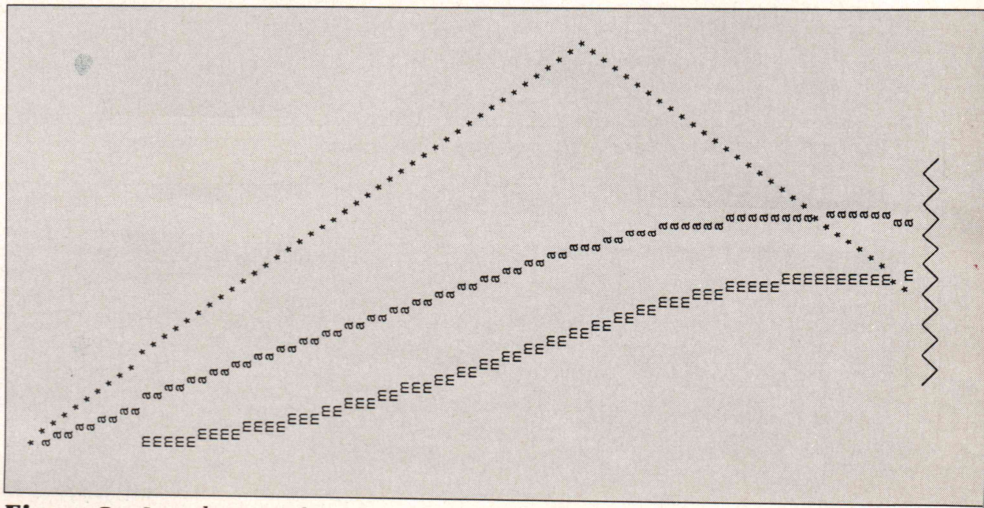


Figure 3c: *Length-64 window*

boxcar average is converging on the true mean exponentially and reaches it after about 50 points are processed (there are 100 points in the graph).

Looking at Figure 3c, the same thing is happening. The boxcar average is converging gradually on the true mean (toward the right side of the graph). In fact, this characteristic is generally true. A boxcar average will converge on the arithmetic mean given enough input points. The shorter the boxcar, the faster it converges. Note, however, that if the boxcar is too short, you end up with a digital filter that never converges, as in Figure 2a. Here the boxcar just tracks the input data without ever approaching the mean. So you have to do a balancing act. The longer the boxcar, the closer you'll get to the

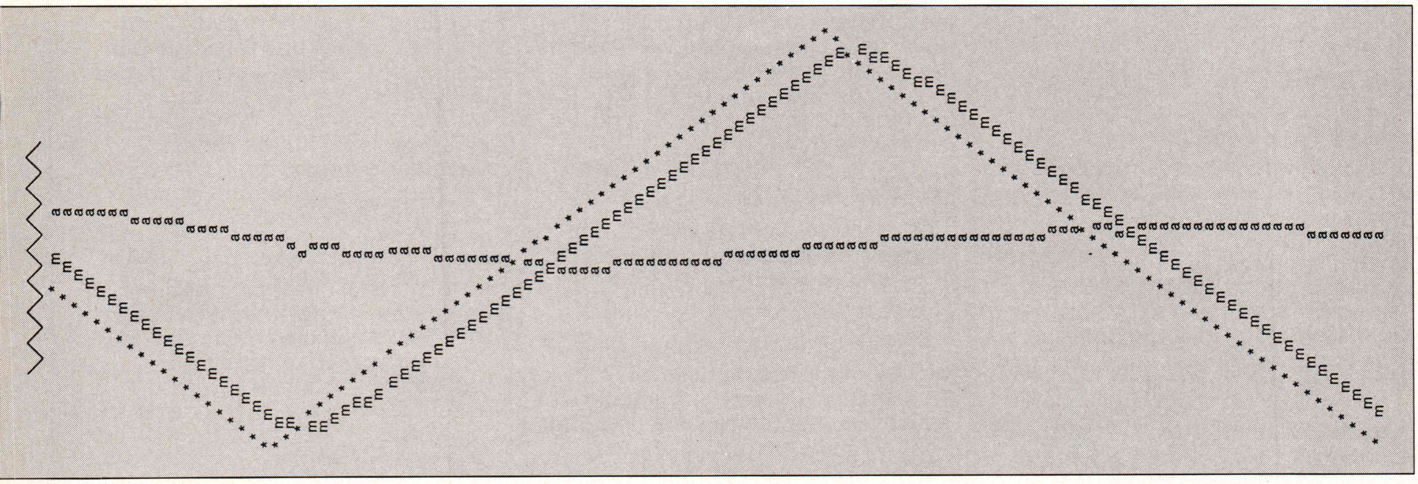
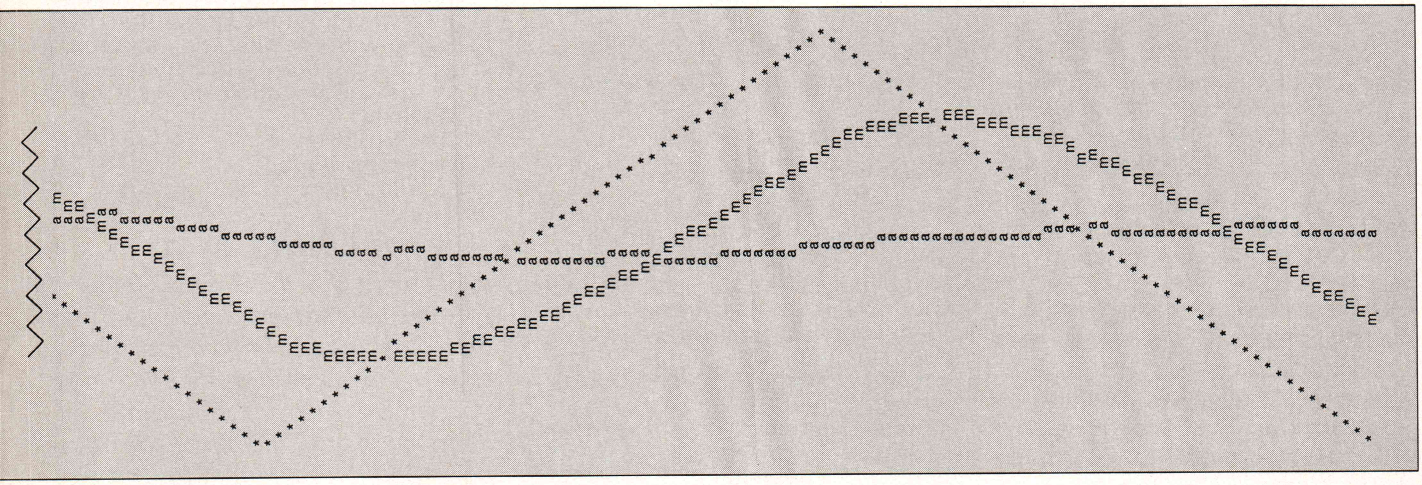
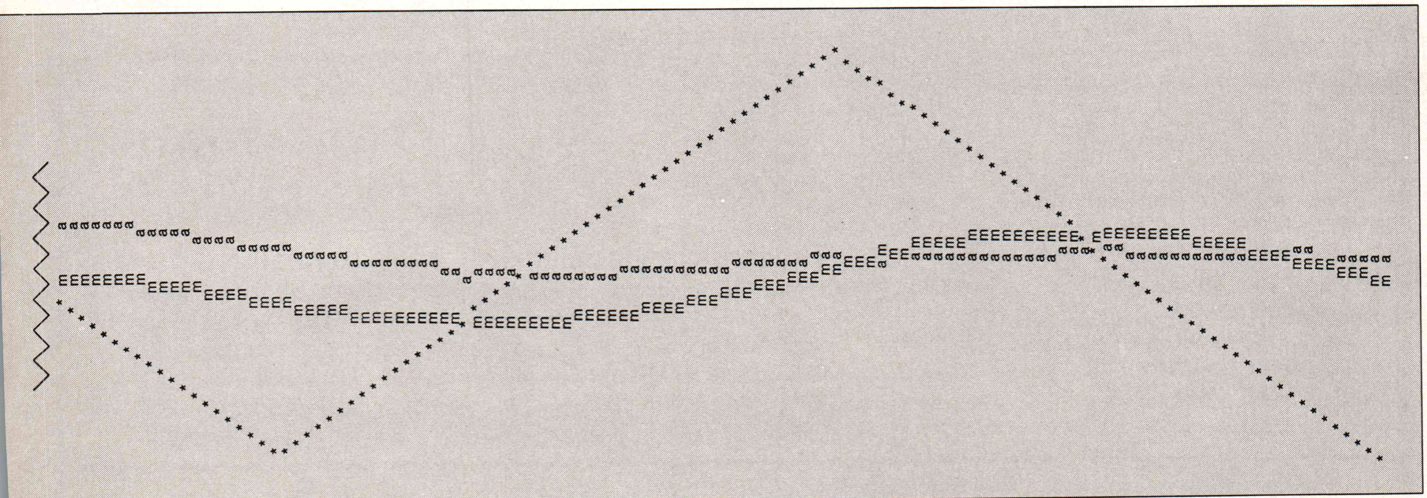


Figure 3a (above): *Length-4 window*





C CHEST

(continued from page 107)

true arithmetic mean. On the other hand, a long boxcar takes more time to converge than a short one. If the boxcar is too short, you'll never converge.

Figure 4, page 110, shows the boxcar algorithm being applied to a set of input points randomly distributed around a straight line. As before, asterisks are used to mark the points, sideways *a*s are the arithmetic mean, and sideways *m*s are the boxcar average. Figure 4a shows a length-4 boxcar. Here the boxcar output jumps around almost as much as the input data, so it's not all that useful. In Figure 4b, a length-16 boxcar is applied. The boxcar converges on the mean after about 50 points. It still jumps around a bit, though. Figure 4c shows a length-64 boxcar. It tracks the arithmetic mean very closely after about 150 samples. On the other hand, it takes 150 samples to get close enough to be useful. Note that this last example could be made to converge faster if you initialized the boxcar to the arithmetic mean of the first few points rather than to zero.

Implementation

All this stuff is implemented by the short set of routines in Listing One, page 97. In fact, Figures 3 and 4 are output from the program in Listing One. There are six subroutines here. *Newsample()* passes a new sample into the boxcar. It is called for every input point. *Running_mean()* returns the current value of the exponential boxcar average, *true_mean()* returns a true arithmetic mean, and *deviation()* returns an approximation of the standard deviation (also computed with an exponential boxcar). Given the distance (call it D) between any given sample and the true mean, the standard deviation is the square root of the average D^2.

The *true_mean()* function is here mostly to check the algorithm. It won't work if the sum of the input samples requires more precision than a *long* can muster.

The boxcar average is calculated on lines 39–41 of Listing One, using the algorithm described earlier. The standard deviation is computed in a similar way. A boxcar average of the squares of the differences between the boxcar mean and the current input sample is kept by the code on lines 43–48. The difference is figured on line 43. It's squared on line 44 and then added into the running average on lines 46–48. The square root is taken on line 71, when the standard deviation is requested. This delay saves you the overhead of taking a square root with every sample. On the down side, you can overflow the boxcar if the samples are too big.

The final subroutine of interest is *reset_mean()* on lines 76–85. It resets all the boxcars to 0, and its parameter can be used to set the boxcar length. Note that because *Boxlen* is used to do a right shift rather than a divide, the parameter to *reset_mean()* is actually 2, raised to the *boxcar_val*th power. You may want to add a second argument to this routine—an initial value of the boxcar. You can then take the *true_mean()* of the first few samples and use that value to initialize the running mean.

Exec( ) *functions in the Microsoft C compiler don't work correctly when* putenv( ) *is also used.*

You may want to make several other changes, depending on your application. First, because the true mean is not all that reliable, you'll probably just want to remove it from the routines. Delete the *Average* and *Numnums* variables and all references to them, including the *true_mean()* subroutine. Next, all my variables are *unsigned longs*. Consequently, I can't keep a fractional mean around, and the range of the data is limited to the precision of a *long*. You may want to change all these to *doubles*. Finally, *Boxlen* is the number of bits to shift rather than a true divisor. This means that the boxcar length is limited to powers of 2, which may not be enough resolution for you. On the other hand, it lets you use an efficient implementation of the boxcar. If you go to something other than a power of 2, you'll need to use something closer to the naive algorithm, introducing an extra divide into the algorithm and slowing it down. One final easy-to-do improvement helps with the start-up time and was mentioned earlier. As you've seen, when left to its own devices, the algorithm converges exponentially. You can improve this behavior by taking the true mean of the first few samples and then using the value of the true mean as the algorithm's starting point, rather than 0.

Microsoft Bug of the Month

A quick note about a bug I found in the Microsoft C compiler, Version 4.0. The *exec()* functions don't work correctly when *putenv()* is also used in the same program. Memory gets fragmented in ways that make a second *exec()* call return with an "out of core" error status (ENOMEM). One way around this bug is to mimic Unix and write your own *putenv()*. Start out in *main()* by copying the environment strings into a static array. *Envp*, an *argv*-like pointer to the environment strings, is passed as the third argument to *main()*. There's no count, however—*Envp* has a *NULL* in the last entry. Once the environments are copied, you can add new strings to your own static array instead of calling *putenv()*. You can then *exec* to another program with the *execve()* function, which is passed an environment that it in turn passes to the child process. Of course, if you use this method, you'll need to write your own *getenv()*, too (because the default routine doesn't know about your static array).

Nifty Stuff

I get a lot of stuff in the mail. Most of it's not very interesting, but occasionally something useful comes along. Hitherto, I've just mentioned the products I've liked without giving much additional commentary, but I've decided to start writing slightly longer reviews for products I particularly like. Two such are Custom Software Systems' implementation of the Unix vi editor (PC/VI, Version 1.11) and Lattice's version of the Unix

C CHEST
(continued from page 109)

make utility (LMK, Version 2.20b).

PC/VI

PC/VI is a full implementation of vi—and I do mean full. From the user's perspective, it is identical to the Unix program. If you're familiar with the real vi, you can install PC/VI and then use it immediately, without ever looking at the manual. Unlike other vi implementations I've seen, PC/VI supports full visual and ex modes, macros (using both *:map* and *:abbreviate*), and full regular-expressions (in both searches and ex-mode substitutions), and it can edit very large files (though it slows down when the file gets too large). It even has a LISP

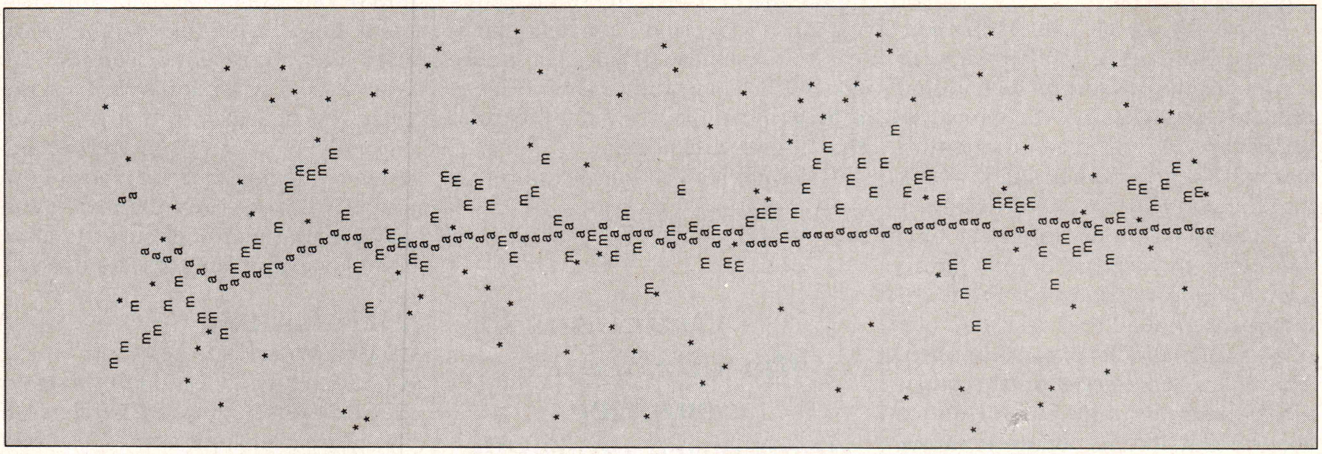


Figure 4: *The effect of exponential smoothing on randomly distributed data.* **Figure 4a (above):** *Length-4 window*

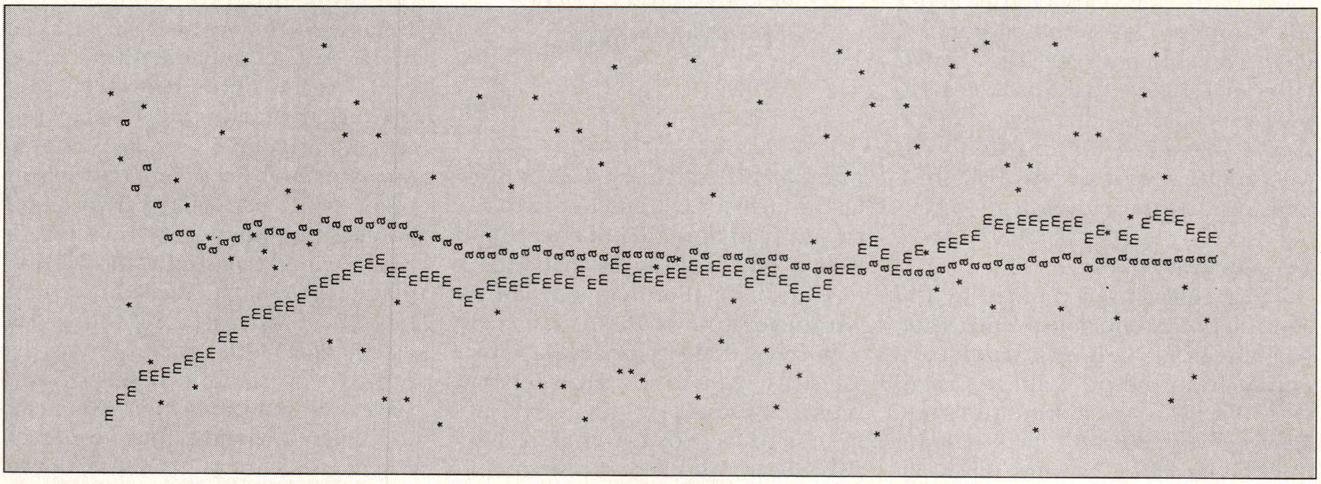


Figure 4b: *Length-16 window*

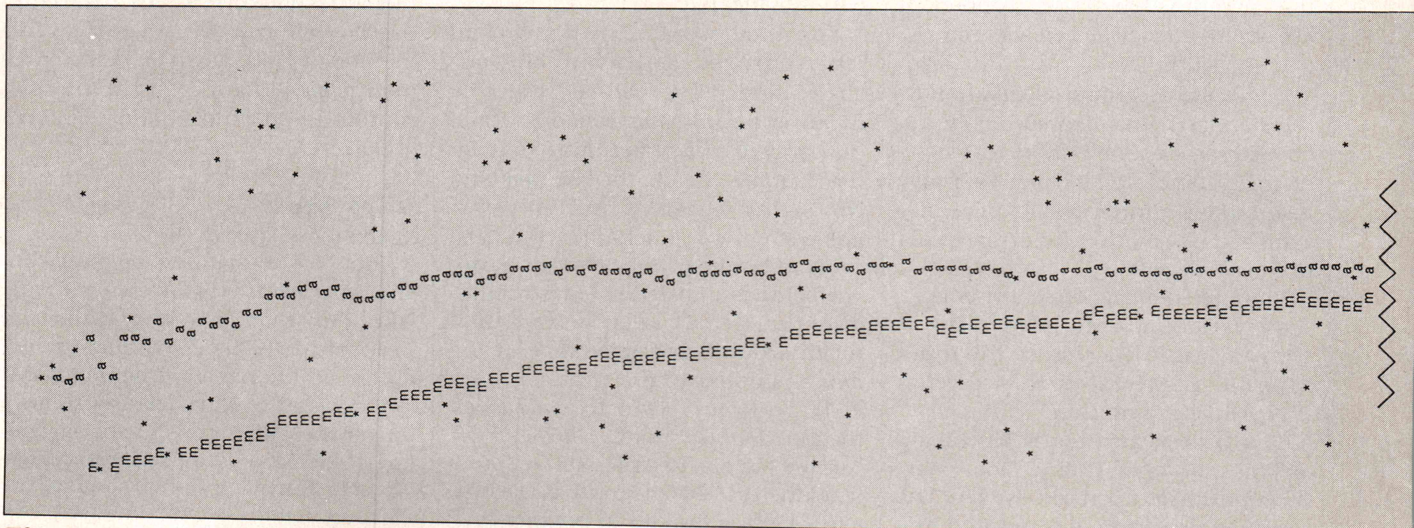


Figure 4c: *Length-64 window*

mode. Moreover, Version 1.11 is bug free as far as I can tell. (Z, the vi shipped with the Aztec compiler, has an annoying tendency to go off into outer space occasionally, taking your work with it.) The PC/VI shell escape works perfectly with both my own shell and COMMAND.COM; it gets the shell's name from COMSPEC. Because Version 1.11 uses unique file names for temporary files, you can even invoke PC/VI from within a shell that was created from within PC/VI—assuming you've enough memory. It supports all the vi command-line switches (*+number*, *+/pattern*, *-ttag*, and so forth) and an EXINIT environment too.

PC/VI also supports tags, a feature particularly useful to programmers. Tags give you the ability to edit a large program by subroutine rather than by file. You first run a program called CTAGS, which creates a tags file, a cross-reference of your C program. You can then ask PC/VI to show you a particular subroutine (either with a *:ta<file>* command or with a Ctrl-]), and it will automatically save the current file, read in the file that holds the subroutine, and position the cursor at the first line of the subroutine itself. PC/VI comes with a CTAGS, though this program works only with C and assembly-language source files. It's not too difficult to make your own tags file using a program such as grep, however.

And if all this weren't enough, PC/VI is terminal independent—it's not tied down to the IBM PC. It uses TERMCAP files for configuration purposes and even comes with a complete TERMCAP interface library that you can use in your own programs (it's an object-module library that can link only to Microsoft C compiled programs, though).

Several versions of PC/VI are on the distribution disk (for a vanilla PC, for a non-PC MS-DOS computer, for a PC/AT, and so forth), so you can choose the version that's most appropriate for your application. It also comes with CTAGS; the TERMCAP libraries, a TERMCAP configuration file with support for about 15 terminals (IBM PC, VT-100, ANSI, H-19, D4XX, and so forth as well as support for Hersey Micro Consulting's FANSI-CONSOLE driver); and SPLIT, a utility that breaks up large files into smaller chunks. PC/VI does have one failing—it doesn't support the *!!* command from visual mode (which executes a command from a subshell and then inserts the standard output from that command directly into the document). This can be accomplished by redirecting to a file from a normal *:!* shell escape and then reading the file with a *:r*, however.

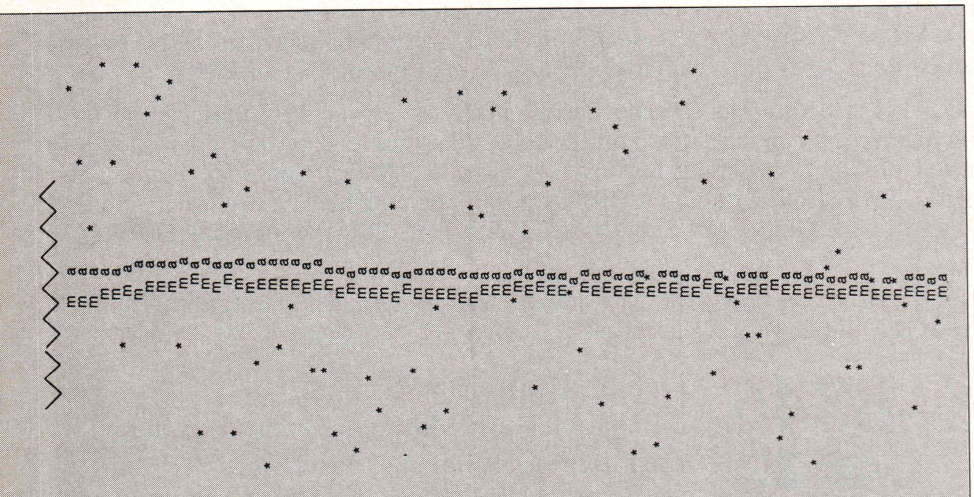
Make

The Lattice make, called LMK, is an enhanced version of the Unix make utility. It has all the functionality of the Unix program, and to my knowledge, it's the most powerful on the market (though, at $125, it's also one of the most expensive). The only make that comes close is Polymake (from Polytron Software), but Polymake has two limitations that grate after a while. I should preface my complaints by saying that the version of Polymake I've been using is more than a year old—I haven't had any notifications of an update, though.

The first problem with Polymake is its inability to deal with subdirectories correctly—at least, it gets very confused when the dependencies, the file being made, and the makefile itself are all in different directories. The ability to deal with subdirectories is useful when you keep a large library directory and you want the sources in one subdirectory, the object modules in a different subdirectory, and the library itself somewhere else again. LMK has none of Polymake's limitations; it happily makes anything anywhere.

The second problem with Polymake is the large amount of memory it uses by itself when it's working. I need this memory for my own programs. Of the three makes I have around, Polymake performs the worst in this department (requiring almost 96K for itself); next (at 84K) is the make that comes with the Microsoft C compiler. The best performer, however, is LMK, which uses only 35K of core. (I got these numbers by running chkdsk from within a makefile.) A lot of memory is saved by the Lattice product's simply not executing a shell unless it actually has to. If you tell LMK to run a normal program, it does so without the extra baggage of a second command interpreter in memory.

I should add as an aside that I'm using my own shell, rather than COMMAND.COM, as a subshell to LMK. It has no problems with this configuration as it reads the shell's name from the COMSPEC environment rather than assuming that the shell is called COMMAND.COM. This capability lets me use my own shell's script files from within make (rather than normal COMMAND.COM batch files). Unfortunately there's one unnecessary, shell-related problem—LMK wastes memory by invoking a shell to do redirection. It ought to do the redirection itself.



Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City,

Ever Program On A Silver Platter??

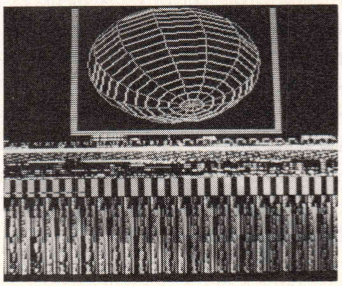
How much would you expect to pay for a 32 bit MC 68000 computer that's a mainframe condensed down into a keyboard? How about $188.00!!?? If it makes you feel any better simply add a zero to the price when you order! But that's actually our price!!! The most powerful computer money can ever buy is now the most inexpensive computer money can buy!!! So don't buy the name! Buy the power!! The power is **not** in the name!

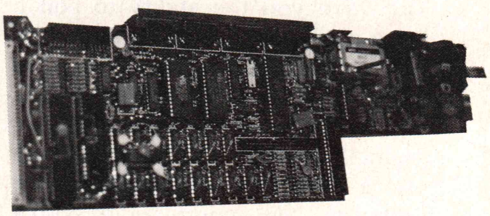
If **you** had the opportunity to work amongst Machine Code ROM Designers, VAX & UNIX wizards in a research laboratory, designing an MC 68000 based computer that's 2nd to none. . .

What would you come up with?? And what would you call it??

Well It's Already Been Done!!
They Called It The QL For The Quantum Leap It Is!!

Absolutely a Quantum Leap beyond what you know & use – and it's truly like Programming on a Silver Platter!!





The QL Desktop Minicomputer: Designed by SRL Labs, manufactured by Samsung. An absolute Quantum Leap beyond all the rest! The phenomenal open architecture QDOS: with Virtual Memory RAM, Multitasking Job Control, Multiuser Networking. It'll Cache Files into unused Memory and create/delete Directories Automatically! Even allows File Names up to 36 characters long! Everything is built into ROM here: QDOS, Networking, Windowing, & 32 Bit SuperBasic, all in a totally concurrent non-destructive environment. Unlimited quantities & lengths allowed with: Variables, Program Lines, CONsoles & Buffers. Dynamic non-destructive virtual RAM Disking & Networking buffers too! Even a System Variables Brain Page Screen! Built-in DCE & DTE Serial Ports.

Language Environments:
Metacomco's "C", LISP, BCPL, 68000 Assembler, APL, Development Kits. Prospero's Pro Pascal & Pro Fortran 77. Digital Precision's Forth-83. QJUMP's 65C02 or 8088 Cross Assembly ROMs. Everything generates native 68000 Compiled Code. ROM Firmware & Software Package is now available which will even bring it up in CPM!

Imagine working with a 32 bit SuperBasic that's structured like Turbo Pascal, powered beyond PIC Basic, in an interpretor always present with QDOS, all concurrently running in a built-in UNIX-like multitasking job controlled environment with access to 360 fully channeled windows, devices & files by EACH job! 3 Major Compilers already exist for the SuperBasic source alone! TURBO, SUPERCHARGE, QLIBERATOR! The compiled SuperBasic code or ANY other language will multitask and control with QDOS and SuperBasic. The list of ALL the Superior Features would fill this entire publication!

The QL comes bundled WITH PSION Integrated Word Processor, Spreadsheet, Database and Presentation Graphics Programs. PLUS: Our FREEWARE Demos & Utilities with all purchases! Plus $12 Shipping and Handling

Call: (201) 328–8846

QLine BBS: 328–2919

MasterCard VISA

Technical Info & Assistance – –
Our Phones are Manned 24 Hours a Day
Lot Purchases Available. We Direct Distribute.

Quantum Computing, Box 1280, Dover, NJ 07801

CIRCLE 144 ON READER SERVICE CARD

(continued from page 111)

CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

Bibliography

Chamberlin, Hal. *Musical Applications of Microprocessors*. 2d ed. Hasbrouck Heights, N.J.: Hayden, 1985. This is a great introduction to electronic music in general. Chapter 14 has a lucid description of digital filter theory that's understandable to nonmathematicians.

Programs for Digital Signal Processing. New York: IEEE Press, 1979. This book has a wealth of FORTRAN programs for digital signal processing, including filter programs. It assumes you know enough theory to understand the programs, however—it's pretty dense.

Electronotes Newsletter, 1 Pheasant Lane, Ithaca, NY 14850, is a small but meaty periodical for electronic music hackers—people who want to actually build the stuff as well as play it.

Flotsam and Jetsam

Books

Several people have written requesting a reading list that covers both books about C and books about programming generally. This month's Flotsam and Jetsam contains such a list.

First, if you don't know a structured programming language, you're better off starting out by learning Pascal rather than C. Though Pascal isn't as powerful a language as C, it holds your hand quite a bit and forces you into good programming practices. A very good introduction to Pascal is Doug Cooper and Mike Clancy's *Oh! Pascal* (New York: Norton, 1982). Also of interest if you're coming to C from FORTRAN is Brian Kernighan and P. J. Plauger's *The Elements of Programming Style*, 2d ed. (New York: Yourdon, 1978). This book teaches structured programming techniques entirely in FORTRAN, an inherently unstructured language. You'll also need to know a little assembly language to learn C. In the IBM PC world, a good introduction to 8086 assembly language is Robert Lafore's *Assembly Language Primer for the IBM PC and XT* (New York: Plume/Waite, 1984). The rudiments of assembly language are also covered in *The C Companion*, discussed later.

In addition to programming languages, you'll need to know about data structures—binary trees, queues, linked lists, hash tables, and the like. Two excellent books are Robert L. Kruse's *Data Structures and Program Design* (Englewood Cliffs, N.J.: Prentice-Hall, 1984) and Aaron M. Tenenbaum and Moshe J. Augenstein's *Data Structures Using Pascal*, 2d ed. (Englewood Cliffs, N.J.: Prentice-Hall, 1986). Both books contain extensive examples written in Pascal. Kruse's explanations (and his programs) are a little more clear than Tenenbaum and Augenstein's, and I prefer his book for this reason. Tenenbaum and Augenstein's book is more comprehensive, however. Also of interest is Robert Sedgewick's *Algorithms* (Reading, Mass.: Addison-Wesley, 1983), which contains algorithms for doing just about everything imaginable (at least in the realm of programming). Everything from splines to Gaussian elimination to sorting routines to fast Fourier transforms is covered. It's an invaluable reference.

As for C itself, the C language was originally brought to the world's attention by Brian Kernighan and Dennis Ritchie in *The C Programming Language* (Englewood Cliffs, N.J.: Prentice-Hall, 1978). The book is usually called K & R. Unfortunately, this book is dense to the point of unreadability in places. I don't recommend it unless you're a very experienced programmer, but if you fall into that category, it's very good. K & R is just a language description; it assumes that you know how to program. The best general introduction to C that I know of is Bryan Costales' *C from A to Z* (Englewood Cliffs, N.J.: Prentice-Hall, 1985). Herbert Schildt's *C Made Easy* (Berkeley, Calif.: Osborne/McGraw-Hill, 1985) is also good. Both of these books are much more readable than K & R. Neither covers the advanced parts of the language in depth, however.

There are several nontextbooks that are good aids to learning C. My own book *The C Companion* (Englewood Cliffs, N.J.: Prentice-Hall, 1986) was developed as supplementary class notes for a C class I teach. It covers many of the topics that are left out of most C textbooks—both basic topics such as binary arithmetic and assembly language and advanced topics such as the complex uses of pointers and writing subroutines with a variable number of arguments. Another book worth having is Rex Jaeshcke's *Solutions in C* (Reading, Mass.: Addison-Wesley, 1986). Rex's book is a collection of C programming tips. He explains many of the more advanced parts of the language with numerous short examples.

Two good books of exercises are available. Clovis L. Tondo and Scott E. Gimple's *The C Answer Book* (Englewood Cliffs, N.J.: Prentice-Hall, 1985) contains answers for all the exercises in K & R. It's quite useful if you're using that text. Another good exercise book is Alan R. Feuer's *The C Puzzle Book* (Englewood Cliffs, N.J.: Prentice-Hall, 1982). The problems in this book address virtually every aspect of the C language. Moreover, the problems are designed to familiarize you with common errors that will probably show up as bugs in your programs. Every exercise is accompanied by a detailed solution. I could have saved myself weeks of debugging time had this book been available when I first learned the language.

The best reference to the C language is Samuel P. Harbison and Guy L. Steele, Jr.'s *C: A Reference Manual* (Englewood Cliffs, N.J.: Prentice-Hall, 1984). The second edition (1987), which incorporates the ANSI extensions, should be available by the time you read this column. Of course, the ANSI standard itself (X3-J11) is a good reference. The public review period for the draft standard ended March 7, so the real standard should be available any time now. For more information, contact the X3 Secretariat:

It's invaluable if you're in that category.

DDJ

(Listings begin on page 64.)

Vote for your favorite feature/article. Circle Reader Service **No. 5**.

Computer and Business Equipment Manufacturers Association, 311 First St. NW, Ste. 500, Washington, D.C. 20001-2178; (202) 737-8888. Unfortunately, the standard is very expensive ($65 for a few hundred Xeroxed pages). Check the computer science library of a local university before acquiring a copy for yourself.

The final category of books is the useful-examples-of-nontrivial-programs category. Several good books are available. Joe Campbell's *Crafting C Tools for the IBM PC* (Englewood Cliffs, N.J.: Prentice-Hall, 1986) is packed with useful subroutines and programs for the IBM environment. It is distinguished by numerous Notes on C Usage sections that discuss the C programming issues involved in the programs themselves. William J. Hunt's *The C Toolbox* (Reading, Mass.: Addison-Wesley, 1985) is also packed with useful stuff, including a complete B-tree database management package. It's not as tied into the IBM PC as is Campbell's book. Both of the operating system design books that were reviewed in the December *DDJ* —Ted J. Biggerstaff's *Systems Software Tools* (Englewood Cliffs, N.J.: Prentice-Hall, 1986) and Douglas Comer's *Operating System Design, the Xinu Approach* (Englewood Cliffs, N.J.: Prentice-Hall, 1984)—are good resources. And of course, *DDJ* itself sells the source code for several large C programs—such as my own MS-DOS shell—on disk.

So, I've just skimmed the surface of what's available and have already spent several hundred dollars of your hard-earned money. Hopefully, you'll find more extensive reviews of some of these books in future issues of *DDJ*, but until then you'll at least have a place to start when you go into a bookstore or library. Good luck.

YOUR SYSTEM'S KEY COMPONENT

The Only Magazine By And For Advanced Micro Users.

At last there is a magazine that brings you the strictly technical but practical information you need to stay up-to-date with the ever changing microcomputer technology . . . *Micro/Systems Journal*. *Micro/Systems Journal* is written with the needs of the systems integrator in mind—the individual who's involved in putting together the hardware and software pieces of the microcomputer puzzle.

In each issue of *Micro/Systems Journal* you'll find such useful and progressive articles as:

- Interfacing to Microsoft Windows
- Unix on the PC
- 80386 Programming
- High Resolution PC Graphics
- Using 80286 Protected Mode
- Multiprocessing and Multitasking

You'll get the hands-on, nuts and bolts information, insight and techniques that *Micro/Systems Journal* is famous for . . . in-depth tutorials, reviews, hints . . . the latest information on computer integration, networks and multi-tasking, languages, and operating systems . . . hard-hitting reviews.

To start your subscription to *Micro/Systems Journal*, simply fill out one of the attached cards or write to *Micro/Systems Journal*, 501 Galveston Dr., Redwood City, CA 94063. You'll receive a full year (6 issues) of *Micro/Systems Journal* for just $20, and enjoy the convenience of having M/SJ delivered to your doorstep each month. Don't wait . . . subscribe today!

$5 SAVINGS

Yes! I want to subscribe to

Micro/Systems Journal™

and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______________________ Exp. date ____________

Signature ______________________________________

Name ______________________________________

Address ______________________________________

City ______________________ State ________ Zip ____________

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc.

3012

$5 SAVINGS

Yes! I want to subscribe to

Micro/Systems Journal™

and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______________________ Exp. date ____________

Signature ______________________________________

Name ______________________________________

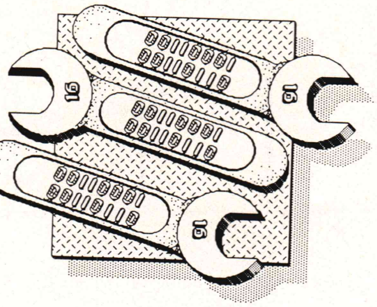
Address ______________________________________

City ______________________ State ________ Zip ____________

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc.

3012



Quality of Life Tools

Today's software market is rife with "user-friendly" MS-DOS shells such as the Norton Commander (which incidentally was written by John Socha, not Peter Norton), X-Tree, and KeepTrack. These little rascals allow naive users to navigate through the hierarchical directory structure, copy files, initialize disks, and the like as long as they can find the arrow keys. I suspect that most experienced programmers stay as far away from these programs as I do—user interfaces that clutter up the screen and constantly ask "Are you sure?" just get in the way during the day-long cycles of edit, compile, link, and debug.

Of course, the default MS-DOS command processor (COMMAND.COM) is a long way from the last word in user interfaces, too, and there is certainly a place in programmers' hearts for new shells that can enhance that interface without altering it past recognition. I'd like to discuss briefly two such products this month and solicit information on other such products from *DDJ* readers and software vendors.

Command Plus

Command Plus is an alternative shell from ESP Software Systems that completely replaces MS-DOS' COMMAND.COM. Command Plus offers significantly enhanced *COPY*, *DEL*, and *DIR* commands and adds a high-performance file *BROWSE* command, com-

by Ray Duncan

mand aliasing, the ability to accept multiple commands on the same line (separated by & delimiters), command logging, and a directory stack. It also includes an extensive shell programming language called SCRIPT that is upward compatible from the normal MS-DOS batch file commands. SCRIPT supports integer, long integer, and string variables; various operations on those variables; sophisticated control structures, such as *CASE*; keyboard input into a string variable; cursor positioning; and a bevy of operations on environment variables and file names, extensions, dates, and attributes.

But for me, the two most useful features of Command Plus are its support for the command history and for regular expressions. Plain vanilla MS-DOS allows you to specify sets of files with the wildcard characters * and *?*. Command Plus' regular expressions give you much more flexible control over file names, with the ability to specify or exclude single characters or ranges of characters in any position. The file specification *[abcd4–9]*.asm*, for example, matches any file with the extension *.asm* and whose name begins with one of the letters *a* through *d* or with one of the numerals *4* through *9*.

The history feature of Command Plus pushes each command onto a history list as it is entered. The default size of this list is 10 commands (the oldest one is simply lost as each new command is entered), but you can configure it to hold as many as 48 commands.

You can recall and display a previous command by using the arrow keys to traverse the list, edit the command if necessary, then press the Enter key to carry out the command again. For example, if four commands ago you entered:

```
LINK FOO,,,SLIBW+LIBH,FOO.DEF
```

and you want to repeat the same link with a different library, you would just hit the up arrow four times to redisplay the original *LINK* command, move to the position to be changed with the left and right arrow keys, type the new library name, and hit Enter. Incidentally, more editing functions are available than in normal MS-DOS (such as word tabbing right or left), and the editing keys are configurable. You can also view the entire history list and select old commands for editing by their number on the list if you wish. I have found that the history feature saves hundreds of keystrokes and mistyped commands daily.

Now, I know Unix partisans are going to write in and tell me once again that Unix shells have had these features since the Dark Ages. I am aware of this already, I too am pleased that Unix has at least one or two redeeming features, and I'll be the first to agree that Unix software comes from the Dark Ages. But it's nice to have histories, regular expressions, and a decent script language without having to sacrifice a megabyte of RAM, 10 megabytes of fixed disk, and half my CPU cycles on the altar of Unix.

The only significant deficit I see in the current release of Command Plus is the lack of support for replaceable parameters in command aliases. We can always hope that the vendor will see fit to add this in a future version! You can obtain more information on Command Plus from ESP Software Systems Inc., 11965 Venice Blvd., Ste. 309, Los Angeles, CA 90066; (213) 306-7408.

ProCED

ProCED, written by Chris Dunford, is an extremely powerful command-line editor for MS-DOS. It is loaded as a Terminate and Stay Resident (TSR) utility and is not a complete replacement for COMMAND.COM. What it

does offer is vastly improved command-line editing, support for synonyms (aliases) with replaceable parameters, command chaining, command logging, and inspection/recall/editing of previous commands via a history list similar to that described earlier for Command Plus (note that ProCED predates Command Plus by a considerable margin, however). Nearly all ProCED's special characters, buffers, and stacks are configurable by the user, and lists of synonyms and other configuration information can be automatically loaded from a file at system start-up.

ProCED works by capturing and replacing the MS-DOS buffered keyboard input function (*int 21h*, function *0ah*); therefore, its capabilities are available in any MS-DOS program that performs its input through this function, including most debuggers. By the same token it does not work with some programs, such as the Norton Commander, that perform their keyboard input character by character.

A particularly nice aspect of ProCED is that it contains "hooks" that allow programmers to write and install new memory-resident commands, called "user commands." These are loaded under the control of ProCED-like miniature TSRs and behave as though they were COMMAND.COM "intrinsic" commands. The ProCED package includes several examples, including *ATTRIB* (displays or alters file attributes), *CDIR* (a sorted directory), and *SEND* (transmits an arbitrary string of data to any file or device).

Chris recently sent me a development (prerelease) version of ProCED that has two terrific new features. The first is called command extrapolation: if you type a few letters of a command and then press Ctrl-X (^X), ProCED searches the history buffer for the first matching command and displays it for editing. If it doesn't find the one you want, you simply hit ^X repeatedly until the command you desire appears. The second new feature is a built-in file-name search, which is similar to command extrapolation.

At any point in a command line, you can type a partial file specification, or a file name that contains wildcards, and then press the Tab

(continued from page 119)

key to have ProCED replace the tentative file spec with matches from the current directory. You can also move to a new directory by pressing the space bar when a directory name appears and then press the Tab key again to see further file names. Command extrapolation and file-name search will be present in the next retail version of ProCED, which may be available by the time you read this.

Time for a testimonial: I have been using ProCED for at least a year and wouldn't want to live without it. I even carry a copy of it with me on a floppy disk when traveling, so I won't feel deprived when using someone else's PC! ProCED is available from the Cove Software Group, P.O. Box 1072, Columbia, MD 21044; (301) 992-9371.

80286 Resources

In a recent column, I mentioned the book *Inside the 80286* (by Ed Strauss, published by Waite Group/Brady) as an excellent source of information on the 80286's protected mode, virtual memory management, task switching, and interrupt handling. *DDJ* readers have also brought to my attention the following book:

Morse, Stephen P.; and Albert, Douglas J. *The 80286 Architecture*. New York: Wiley, 1986. 279 pages including index. ISBN 0-471-83185-9.

The architectural and systems-level discussion that occupies nearly all of Strauss' book is compressed into about 40 pages here. Most of the remainder of the book is a primer on the 80286's opcodes and addressing modes, including the traditional explanation of bits, bytes, and hex arithmetic and some rehashed tables from Intel manuals. The last chapter of the book is entitled "286 Hardware: Building a Computer," but its discussion is generic and somewhat vague whereas in Strauss' book the hardware discussion is extremely specific and includes schematics and software listings that you could use to breadboard your own primitive 80286 machine. All in all, although Morse and Albert's credentials are good, I don't think this book makes the grade. I'd recommend you spend your money on Strauss' book and a copy of the Intel *80286 Programmer's Reference* instead.

I have been using ProCED for a year, and I wouldn't want to live without it.

Pretty Pictures Department

I suspect at least half of the readers of this column reacted to the *Scientific American* article on Mandelbrot sets by writing their own programs to plot these intriguing images. Most of us, however, also found that plots in the 350-line, 16-color modes of the EGA take nearly forever unless the program is painstakingly optimized. Relax, someone else has done the work! A slick, fast program to plot Mandelbrot images, called Fractal-Magic-EGA, is available for $25; the source code (Turbo Pascal) is also available for an additional $50. Contact Sintar Software, P.O. Box 3746, Bellevue, WA 98009; (206) 455-4130.

Performing SETs from a Program

Daniel Briggs of the Solar Astronomy Department at CalTech, writes: "Someone recently wanted to know about how to change directories in a batch file and then return home again. There are several different approaches, but here's the one I took.

"The GETDIR.ASM program [Exam-

ple 1, right] illustrates a means of providing string functions to batch files. In this case, the program *SET*s the value of the environment variable *DIR* to be the current default directory. This value can then be referenced later in a batch file in the form *%DIR%*. The undocumented hook into the current copy of the command processor, *int 2eh*, is used to do the *SET*—somewhat unportable but by far the best method among the various alternatives. This program can be altered easily for other uses by changing the value at *var_name* to the desired variable name and the function *get_value* to the desired string function.

"With the GETDIR program in hand, you can write sequences such as this in batch files:

get-dir ! equivalent to "set DIR=<current directory>"
cd <somewhere else>
<do stuff>
cd %DIR% ! %<variable name>% replaced by its value

"The GETDIR approach has the disadvantage that it cannot be nested. There is a set of shareware utilities out that includes the programs PUSHDIR and POPDIR, which do what they sound as though they do. Try a local BBS; they're fairly common. I like my approach, though, because it provides a simple means of adding any string function that you can dream up into a batch file."

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and disk format (MS-DOS, Macintosh, Kaypro).

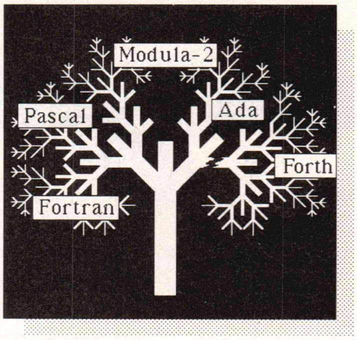
DDJ


```
; GETDIR.ASM --- set environment variable DIR - current directory path
;
; Copyright (C) 1986 Daniel Briggs
;
; This program illustrates a means of providing string functions
; to batch files.  In this case, the program will set the value of
; the environment variable DIR to be the current default directory.
; This value can then be referenced later in a batch file by
; using the phrase %DIR%.  The undocumented hook into the current
; copy of the command processor, int 2eh is used to do the set.
;
; To assemble, link, and convert to an executable COM file:
;
;       MASM GETDIR;
;       LINK GETDIR;
;       EXE2BIN GETDIR.EXE GETDIR.COM
;

stk_size        equ     1000h           ;a good healthy stack

cseg            segment para public 'CODE'
                assume cs:cseg, ds:cseg, ss:cseg, es:cseg

                org     2ch
env_ptr         label   near            ;points to local environment

                org     100h            ;skip to the end of the PSP
DOS_entry       label   far
                jmp     set_str

parameter       db      0, 'set '       ;start of command to be passed
var_name        db      'DIR='          ;counted string name
var_value       db      80 dup (?)      ;buffer for variable value

iSP             dw      offset end_code + stk_size

set_str         proc    near
                cld                     ;addressing already set by loader
                mov     sp, iSP         ;set stack pointer

                mov     si, offset var_value
                call    get_value       ;fill value with desired string

                mov     al, 0
                mov     cx, -1
                mov     di, offset parameter+1
                repne scasb                       ; find the null
                dec     di                        ; point to it
                mov     byte ptr ds:[di], 0dh     ; ascii CR
                not     cx                        ; create count byte
                mov     parameter, cl

                mov     bx, iSP         ;top of program
                mov     cl, 4
                sar     bx, cl
                inc     bx
                mov     ah, 4ah         ;es already set appropriately
                int     21h             ;shrink down to memory needed

                mov     si, offset parameter
                int     2eh             ;invoke the command processor

                push    cs              ;reset stack
                pop     ss
                mov     sp, iSP         ;put the stack pointer back

;
; Can you find a means of getting a status back from this technique?
;
                mov     ah, 4dh
                int     21h             ;get the returned status  <**wrong**>

                mov     ah, 4ch         ;terminate w/status
                int     21h
set_str         endp

comment         /
                This is the routine that actually sets the value of the
                string.  In this example, it sets the buffer pointed to
                by DS:SI to the current pathname.  What ever string is set
                must be terminated by a null.  The procedure
                assumes that the buffer starts initialized to 0s, so does
                not add the final null.
                /

get_value       proc    near
                mov     byte ptr ds:[si], '\'   ;not provided by DOS
                inc     si
                mov     ah, 47h
                mov     dl, 0
                mov     di, si
                int     21h                       ; get current directory
                ret
get_value       endp

end_code        label   near

cseg            ends

                end     DOS_entry
```

***Example 1:** Providing string functions to batch files*

True BASIC Challenges Modula-2

by Namir Clement Shammas



In this issue I will discuss the implementation of modules in Version 2 of True BASIC, present two True BASIC modules, and compare True BASIC's modules with those of Modula-2.

The dichotomous nature of True BASIC is marked by the support of structured code on one hand and the lack of structured data on the other. Modules as well as external libraries seem to add more code structure constructs. The fact that they help create reusable software libraries is welcome as an effective timesaving tool and the implementation of modern software engineering methodology.

When I first heard about modules in True BASIC, I asked the technical staff at True BASIC Inc. how different they were from external libraries. They answered by pointing out that modules offer better control over exported routines and a more advanced data interface. You can access all the routines in a library, and in addition, the local variables in library routines are invisible to other routines both inside and outside the library. This means that the argument list is the main path for sharing data, aside from using temporary data files.

Modules modify the interaction of routines and data by supporting the following sections:

- *MODULE <name>*
- *PUBLIC <list of exported variables and arrays>*
- *PRIVATE <list of unexported routines>*
- *DECLARE DEF <list of functions>*
- *SHARE <list of variables and arrays common among the module routines only>*
- module initialization code
- functions and procedures
- *END MODULE*

The *PUBLIC* declaration is used to list the names of scalar variables and arrays that are exported to client programs. This makes public variables global and accessible to all parts of the module as well as to the program that calls the module.

The *PRIVATE* declaration lists the functions and procedures that are local to the module. Compared to Modula-2, this is the reverse of the *EXPORT* list. All routines that are not listed as private can be called by client programs.

The *DECLARE DEF* statement indicates the names of all the functions defined inside the module.

The *SHARE* declaration lists the scalar variables and arrays that are accessible to all the module routines but not to the client programs. They need not and should not appear in the routines' argument lists. The additional advantages of shared variables are:

1. They support data abstraction by enabling the access of data structures while hiding the details.
2. Shared variables are static: they retain their values between calls to routines in the module.
3. No conflict results from using the same variable names in a client program or shared variables in other routines.

Module initialization is carried out automatically before the program starts running. This means that any *PUBLIC* variable involved in the initialization step must be assigned an initial value from within the module itself, which makes the initialization step independent of the client programs.

The new version of True BASIC has a powerful *LOAD* command, which enables you to load libraries and modules and so extend the True BASIC language. Loading a library that implements hyperbolic functions, for example, enables you to type (in the command mode) *PRINT SINH(2.4)* and obtain a result. The loaded libraries and modules cut down on the time required to link programs with external libraries and modules.

Examples

I will now discuss two examples of True BASIC modules. Example 1, page 125, contains a linear regression module. The *PUBLIC* declaration exports the three statistics slope, intercept, and coefficient of determination. The *PRIVATE* statement declares routines *Mean* and *Sdev* as local to the module. *DECLARE DEF* points out that *Mean* and *Sdev* are local functions used to calculate the mean and standard deviation for intermediate results. The *SHARE* statement lists the statistical summations as shared static variables.

The call to *InitializeSum* is the first statement in the module initialization section. It is followed by assigning numeric codes for missing data to the three regression statistics. These values can be used by the client program to detect that no meaningful results are available and so distinguish them from random junk data.

The *InitializeSum* subroutine is used to zero the statistical summations—this is carried out automatically when the program starts running. For the sake of clarity, I called the subroutine from the program. Subroutine *AcumData* takes data from

the two arrays *X* and *Y* and updates the summations. Your program can call it repeatedly to process data in batches. Since the summations are static variables, they maintain their values between calls. Thus, when calling *LineFit*, the summations supply the required information to calculate the regression statistics.

Example 2, page 126, shows a simple True BASIC program that uses the regression module. The *LIBRARY* keyword is used to import it, and *DECLARE PUBLIC* is used to import the public variables exported by the regression module. Notice that the statistical summations are invisible to the client program. If I want to write a version that uses an external True BASIC library, I must pass the summations in the argument lists. Calling *LineFit* involves no parameters because I have elected to make the results public.

My second example involves sorting and indexing an array of strings. In writing the program I made certain choices to demonstrate several features of True BASIC modules. Example 3, pages 128–129, shows module *Sort*. The *PUBLIC* statement indicates that the module declares the array *Item$()* and the array counter *NData* as globally accessible to client programs. The *SHARE* declaration lists two arrays: the first is an array of pointers; the second is an index table. These arrays are shared within the routines of the module. The module initialization consists of assigning values to shared variables.

The *Sort* module consists of three exported routines. The first one ensures that the sizes of public array *Item$()* and the shared pointer array are adequate. The *REDIM* statement (one of the new features in True BASIC, Version 2) expands the arrays as needed. The size of the index table is independent of the size of array *Item$()*. It maps indices for the characters *A* through *Z* in uppercase only.

Subroutine *Sort_and_Index* performs two tasks: it sorts the array *Item$()* using pointers and then sets up an index table. It calls the local subroutine *ShellSort* to perform a pointer-based Shell sort on array *Item$()*. The second local subroutine, *Set_Index*, is called to set up array *Table()*. The first entry encountered starting with the letter *A* is stored in location 1 of array *Table()*, that of *B* in location 2, and so on. The table index is initialized with 0s.

Function *Search_Index* is used to search for a specific occurrence of a string and returns the index of array *Item$()* or 0 if not found. Using the index table, this function is able to zoom in on a feasible search range, knowing where to start and stop.

Example 4, page 130, shows a client program that uses module *Sort*. It contains *DATA* statements that supply the array *Item$()* with some keywords from the Pascal language. The *DO . . . WHILE* loop counts the number of items in the *DATA* statements. A dummy string is used, instead of the *Item$()* array, for the *READ* statement to avoid a possible array-bound error. The *RESTORE* statement resets the *DATA* statement pointer. The program calls for sub-

```
MODULE Regress

! Simple linear regression module

PUBLIC Slope, Intercept, Rsqr ! Global variables

PRIVATE Mean, Sdev ! Routines local to module only

DECLARE DEF Mean, Sdev

SHARE Sum, SumX, SumX2, SumY, SumY2, SumXY ! static local variable

!----------------- Initialize module ------------------
CALL InitializeSum

!---------------- module routine definitions ----------

def Missing = -1.0E+200

sub InitializeSum
! Set statistical summations to zero
let Sum, SumX, SumX2 = 0
let SumY, SumY2, SumXY = 0
let Slope, Intercept, Rsqr = Missing

end sub

sub AcumData(X(), Y(), NData)
! Subroutine to accumulate stat summations

FOR I = 1 TO NData
   let Xt = X(I)
   let Yt = Y(I)

   let Sum = Sum + 1
   let SumX = SumX + Xt
   let SumY = SumY + Yt
   let SumX2 = SumX2 + Xt * Xt
   let SumY2 = SumY2 + Yt * Yt
   let SumXY = SumXY + Xt * Yt
NEXT I

end sub

!--------- define internal functions ----------
def Mean(A,B) = A / B
def Sdev(Sum2, Sum, N) = SQR((Sum2 - Sum^2/N)/(N-1))

sub LineFit

let MeanX = Mean(SumX,Sum)
let MeanY = Mean(SumY,Sum)
let SdevX = Sdev(SumX2,SumX,Sum)
let SdevY = Sdev(SumY2,SumY,Sum)
! calculate sought regression results
let Slope = (SumXY - MeanX * MeanY * Sum) / SdevX^2 / (Sum - 1)
let Intercept = MeanY - Slope * MeanX
let Rsqr = (SdevX / SdevY * Slope)^2

end sub

END MODULE
```

Example 1: *True BASIC source code for linear regression module*

```
! PROGRAM Regress demonstrates calling module "regress"

OPTION BASE 1

!--------------- Module declarations -----------
Library "REGRESS.MDL"
DECLARE PUBLIC Slope, Intercept, Rsqr
DIM X(100), Y(100)

let MAX_DATA = 100

CLEAR ! clear screen
DO
   INPUT PROMPT "Enter number of data points " : NData
   PRINT
LOOP UNTIL (NData > 2) AND (NData <= MAX_DATA)

FOR I = 1 TO NData
  PRINT "For data point # ";I
  INPUT PROMPT "    enter X " : X(I)
  INPUT PROMPT "    enter Y " : Y(I)
  PRINT
NEXT I

Call InitializeSum ! initialize stat summations

Call AcumData(X, Y, NData)

Call LineFit

CLEAR
PRINT USING "Rsqr = #.######" : Rsqr
PRINT USING "Slope      = +#.#####^^^^" : Slope
PRINT USING "Intercept = +#.#####^^^^" : Intercept

END
```

Example 2: *True BASIC source code for application program using the regression module in Example 1*

STRUCTURED PROGRAMMING
(continued from page 125)

routine *Set_Up* in the module *Sort*. This ensures that the public array *Item$()* and the pointer array (local to the module) have enough spaces. The *RESTORE* statement is followed by a *FOR . . . NEXT* loop to read the *DATA* statements into array *Item$()*. The module subroutine *Sort_and_Index* is invoked to prepare the index table. I included a *DO . . . UNTIL* loop to enable you to type in the Pascal keywords and find their location in array *Item$()*.

The second example illustrates data hiding by using static shared arrays within the *Sort* module. The array of pointers and index table remain invisible to the client program. The limitation of shared variables is that there can be only one instance of each variable. To use them as arguments, the module must include routines to store, recall, and manage the shared arrays to simulate and handle multiple instances.

True BASIC vs. Modula-2

How do True BASIC modules compare with those of Modula-2? Here are some comparison aspects:

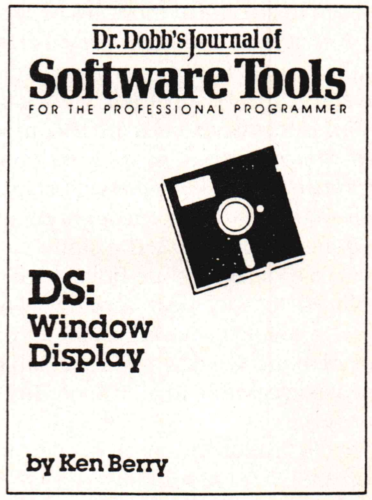
- Using modules in True BASIC is optional. The core implementation has enough constructs to enable you to avoid using modules altogether if you write all the software you use. Attempting the same type of independence is impossible with Modula-2, in which the core language is much smaller and requires you to use modules for common operations such as I/O and string handling.
- In Modula-2, a library module consists of two components: a definition module and an implementation module. Modules in True BASIC are contained in one file. The difference comes from the language design philosophy. Modula-2 is aimed at large advanced software projects involving teams of programmers. In a top-down software project, the specifications of each module must first be determined, hence the need for a definition module that declares module specifications. Modula-2 libraries can be distributed with the source code for the definition module and the symbol, link, and other files for

The Tele Operating System Toolkit

The unique features of this four part, multitasking operating system will allow you to fully exploit the power of any 8086-based machine! **Tele** is written in C and assembly language for IBM PC compatibles and includes **preemptive multitasking capabilities and an unlimited number of tasks.** Tele contains **full C and Assembler source code,** as well as precompiled libraries. It is compatible with MS-DOS, Unix, and the MOSI standard. MS-DOS disk format.

SK: The System Kernel includes the most crucial part of the Tele Operating System - the preemptive multitasking algorithm. **SK** also contains an initialization module, general purpose utility functions for string and character handling, format conversion, terminal support and machine interface, along with a real-time task management system. All other components require SK: The System Kernel.

SK	**Item #090**	**$49.95**



DS: The Display Driver contains BIOS level drivers for a memory- mapped display (the fastest way to display data), window management support and communication coordination between the operator and tasks in a multitasking environment. **DS** includes functions to create and delete virtual displays, and functions to overlay a portion of a virtual display on the physical display.

An unlimited number of virtual displays can belong to any particular task, and an unlimited number can be in the system at any time. Requires SK: The System Kernel.

DS	**Item #091**	**$39.95**

Return this order form with your payment to M&T Books, 501 Galveston Dr., Redwood City, CA 94063
Or, Call **TOLL-FREE 800-533-4372** Mon-Fri 8AM-5PM. In CA call **800-356-2002**.

YES! Please send me:

Item #090 SK: The System Kernel $49.95 ____________
Item #091 DS: The Display Driver $39.95 ____________
Subtotal ____________
CA Residents add tax ____ % ______
Add $2.25 per item for shipping ____________
TOTAL ____________

☐ Check Enclosed. **Make Payable to M&T Publishing, Inc.**

Charge my ☐ VISA ☐ M/C ☐ Amer. Exp

Card # ______________________ Exp. ________

Name ______________________

Address ______________________

City ______________________ State________ Zip ________

3127C

(continued from page 126)

the implementation module. In True BASIC, if you use the run-time package to form .EXE files, you must supply separate documentation. True BASIC module developers can agree on using .DEF files that have routine headings and comments.

- You can include True BASIC modules after the *END* statement of a main program. In Modula-2, modules are always in separate files.
- In True BASIC's *PUBLIC* declaration, variables are exported in a way similar to having variables listed in the Modula-2 *EXPORT* list. True BASIC does not export data types (transparent or opaque), however, because it does not support Pascal-like data structures.
- The *PRIVATE* declaration is needed in True BASIC to specify local routines because they coexist with the exported ones.
- Shared variables in True BASIC modules are somewhat similar to variables in nested local modules in Modula-2 (that is, a module library using a local module). Both types of variables are static and retain their values between successive calls to the modules. Shared variables in True BASIC are more flexible, however, because their scope extends throughout the module. In Modula-2, the static variables in a local module have a scope confined to the local module. This gives True BASIC shared variables the best of both worlds: static variables that are accessible to all the module's routines.
- Module initialization is similar for True BASIC and Modula-2.
- In Modula-2 you are able to resolve the problem of duplicated exported identifiers (variables and routines) more easily. You *IMPORT* the entire modules and use the dot notation to tell the compiler exactly which routine to use. Suppose, for example, that you have obtained a new I/O module (call it *NewIO*) and you want to import the procedure *WriteString()*. At the same time you need to use *WriteString()* from the standard module *InOut*. You simply import both modules using:

```
IMPORT NewIO;
IMPORT InOut;
```

```
MODULE Sort

PUBLIC Item$(100), NData ! Global array and variable

PRIVATE Set_Up, ShellSort ! Routines local to the module

DECLARE DEF Search_Index

SHARE Ptr(100), Table(26), FALSE, TRUE, HI_CHAR, MAX_DATA

!------------------ Module initialization ----------
let TRUE = 1
let FALSE = 0
let HI_CHAR = 26
let MAX_DATA = 100

!---------------- Routines definition ----------------

sub Set_Up
! Make sure that the arrays have enough space
IF NData > MAX_DATA THEN ! adjust array sizes if needed
   MAT REDIM Item$(NData)
   MAT REDIM Ptr(NData)
END IF

end sub

!----------------------------------------------------

sub Set_Index
! Build index table

MAT Table = ZER ! Initialize array

FOR Char_Index = 1 TO HI_CHAR
    let C$ = CHR$(64 + Char_Index) ! --> 'A' to 'Z'
    IF Char_Index = 1 THEN ! Start searching at the beginning
        let Index = 1
    ELSE
        ! Seach backwards
        let Index = 1 ! assume worst case as default
        let J = Char_Index ! use J as copy of index I

      DO WHILE J > 1
          IF Table(J-1) > 0 THEN ! found good 'last index'
              let Index = Table(J-1)
              let J = 0 ! zero to exit loop
          ELSE
              let J = J - 1 ! one step backward
          END IF
      LOOP
  END IF

  let Found = FALSE

  DO WHILE (Index <= NData) AND (Found = FALSE)
      let J = Ptr(Index)
      let S$ = Item$(J)[1:1]
      IF S$ = C$ THEN ! Match found
          let Found = TRUE
          let Table(Char_Index) = Index ! store entry index
      ELSE
          let Index = Index + 1 ! increment index for more search
      END IF
  LOOP
NEXT Char_Index

end sub

!----------------------------------------------------

sub ShellSort
! Sort the pointers and keep Item$() unchanged

! Initialize pointers
FOR I = 1 TO NData
    let Ptr(I) = I
NEXT I

! Start the Shell sort
let Offset = NData
```

(continued on next page)

Example 3: *True BASIC source code for module* Sort

Whenever you want to write a string using the *NewIO* version, you write *NewIO.WriteString()*, and to use that of *InOut*, you write *InOut.WriteString()*. Of course, this forces you to use the dot notation with every identifier imported from *NewIO* and *InOut*. Alternatively, you can use the familiar import list for the most used module and keep using the dot notation for the other ones.

In True BASIC modules, a conflict is present on two levels: public variables and exported routines. True BA-

```
DO WHILE OFFSET > 1
    let Offset = INT(Offset / 2)
    DO
        let InOrder = TRUE
        FOR J = 1 TO (NData - Offset)
            let I = J + Offset
            IF Item$(Ptr(I)) < Item$(Ptr(J)) THEN
                let Tempo = Ptr(I)
                let Ptr(I) = Ptr(J)
                let Ptr(J) = Tempo
                let InOrder = FALSE
            END IF
        NEXT J
    LOOP UNTIL InOrder = TRUE
LOOP

end sub

!-----------------------------------------------------

sub Sort_and_Index

CALL Set_Index
CALL ShellSort

end sub

!-----------------------------------------------------

def Search_Index(Datum$, Occur)
! Search for the n th Occur(ance) of Datum$ in array Item$()
! Use index table for faster search

let S$ = UCASE$(Datum$[1:1]) ! pick first character in Datum$
let Index = Ord(S$) - 64 ! Get index for search table
let Table_Index = Table(Index) ! get index entry
let Occurance = ABS(INT(Occur)) ! assign Occur to local copy

IF Table_Index > 0 THEN ! Yes there is an entry!
    let Found = FALSE
    let More_Loop = TRUE

    DO WHILE (Table_Index <= NData) AND (Found = FALSE) AND
                                        (More_Loop = TRUE)

        let J = Ptr(Table_Index) ! store pointer in J
        IF Datum$ = Item$(J) THEN ! found a match
            let Occurance = Occurance - 1 ! Decrement occurance count
            IF Occurance < 1 THEN ! Yes, this the one we want!
                let Found = TRUE
                let Search_Index = Ptr(Table_Index)
            ELSE ! No! keep searching!
                let Table_Index = Table_Index + 1
            END IF
        ELSE ! Should we keep searching?
            IF S$ = Item$(J)[1:1] THEN  ! Yes
                let Table_Index = Table_Index + 1
            ELSE ! No, we are have gone too far and not found a match
                let More_Loop = FALSE ! stop looping
                let Search_Index = 0  ! search has failed
            END IF
        END IF
    LOOP

ELSE
    let Search_Index = 0
END IF

end def

END MODULE
```

Example 3: *Continued*

STRUCTURED PROGRAMMING

(continued from page 129)

SIC uses the same storage location for identical public variables declared in two or more modules. It is your responsibility to manage the values in duplicated public variables. The solution is relatively simple: reassign their values to program variables with different names. This protects you from unwanted changes in the values of public variables while calling different modules. Concerning duplicate routines, True BASIC considers the first duplicate routines to be valid ones. The only remedy is to change the duplicate routine names.

• All the current IBM PC Modula-2 implementations link entire library modules without performing code optimization. Import one routine, and the rest of the library follows! By contrast, linking a True BASIC main program with libraries and modules is optimized.

The advanced software engineering features of True BASIC add more power and punch to BASIC. The implementation of modules is indeed controversial, and I expect it to generate great likes and dislikes.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal,* 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

Vote for your favorite feature/article. Circle Reader Service **No. 7**

```
! PROGRAM Sort_and_Search

Library "Sort.mdl"
DECLARE DEF Search_Index
DECLARE PUBLIC Item$(), NData

CLEAR

let NData = 0
! Count items in DATA statements
DO WHILE MORE DATA
    let NData = NData + 1
    READ Dummy$ ! use dummy variable
LOOP

RESTORE ! DATA counter
CALL Set_Up ! Adjust Item$() if needed

! Read DATA into Item$(), now that we have enough space
FOR I = 1 TO NData
   READ Item$(I)
NEXT I

CALL Sort_and_Index ! Sort and prepare index table

let Occur = 1 ! Search for first occurance
DO
   INPUT PROMPT "Enter sought keyword or [Q] to exit ? " : Search$
   let Search$ = UCASE$(Search$)
   IF Search$[1:1] <> "Q" THEN
       PRINT Search$;" is item number ";Search_Index(Search$, Occur)
       PRINT
   ELSE
       PRINT
       PRINT "PRESS ANY KEY TO EXIT "
   END IF
LOOP UNTIL Search$[1:1] = "Q"

! DATA statements contain a list of Pascal keywords
DATA WRITE, READ, ASSIGN, SEEK, HI, LO, SQRT
DATA SQR, TAN, SIN, COS
DATA IFF, THEN, ELSE, WHILE, REPEAT, BEGIN
DATA FUNCTION, VAR, TYPE
DATA RECORD, SET, FOR, PROCEDURE, PROGRAM

END
```

***Example 4:** True BASIC source code for application program using module Sort, shown in Example 3*

QUIT DOING GRUNT WORK.

Let Greenleaf do it for you and set you free.

C Program developers, stop slaving!

Greenleaf libraries have the functions you need — already perfected and in use by winning program developers in major corporations such as IBM, EDS and GM.

Between our Greenleaf Functions and Greenleaf Comm Library, we have over 340 functions on the shelf. Each one can save you time and effort. Money, too.

Many C programmers have told us that, even if they only use one or two functions, our products easily pay for themselves:

The Greenleaf Functions

The most complete and mature C language function library for the IBM PC, XT, AT and close compatibles. Our version 3.0 includes over 225 functions — DOS, disk, video, color text and graphics, string, time/date, keyboard, new disk status and Ctr1-Break control functions plus many more!

The Greenleaf Comm Library

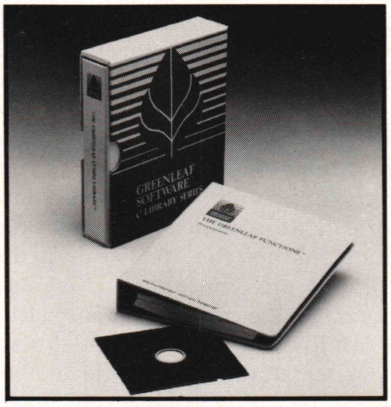
Our 2.0 version is the hottest communications facility of its kind. Over 120 all new functions — ring buffered, interrupt-driven asynchronous communications.

Call Toll Free

1-800-523-9830

In Texas and Alaska, call

214-446-8641



GREENLEAF

Software

Greenleaf Software, Inc.
1411 LeMay Drive Suite 101
Carrollton, TX 75007

If you need more than 2 ports, only Greenleaf gives you the total solution — boards, software, and complete instructions that enable you to build a 16-port communication system.

And no matter how many ports you have, it's virtually impossible to lose information with multiple file transfers. XMODEM, XON/XOFF and Hayes modem controls are featured.

We support all popular C compilers for MS DOS: Lattice, Microsoft, Computer Innovations, Wizard, Aztec, DeSmet and Mark Williams.

Order today!

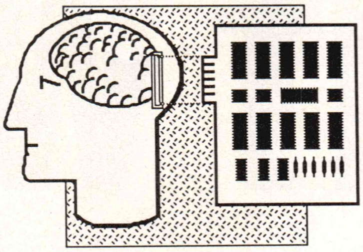
Order a Greenleaf C library now. See your dealer or call 1-800-523-9830. Specify compiler when ordering. Add $8 for UPS second day air, or $5 for ground. Texas residents, add sales tax. Mastercard, VISA, P.O., check, COD. In stock, shipped next day.

Greenleaf Comm Library v2.0	$185
Greenleaf Functions v3.0	$185
Digiboard Comm/4-II	$315
Digiboard Comm/8-II	$515

We also sell compilers, books and combination packages.

CIRCLE 97 ON READER SERVICE CARD

Object-Oriented LISP on PCs



This month I am going to focus on a very powerful yet inexpensive version of LISP for PCs and compatibles that offers an object-oriented extension called SCOOPS and other interesting features.

PC Scheme 2.0

PC Scheme 2.0, from Texas Instruments, is one of the most impressive programming tools currently available for the development of AI-oriented software on PCs. One of the reasons for this is that it is a dialect of LISP compact enough to be viable on the PC without hobbling the power of the language. It is useful as a learning tool as well, both because of its low price and because of its compatibility with the standard textbook used for teaching undergraduate programming courses at MIT—*The Structure and Interpretation of Computer Programs*, by Abelson and Sussman (MIT Press, 1985). PC Scheme has many features that previously have existed only on expensive hardware such as LISP machines. There are limits as to what such features can do when running on small machines, but their very presence is highly welcome and a strong indication of the power now available to programmers at an affordable price.

PC Scheme is a superset of the Scheme dialect of LISP, which was developed ten years ago at MIT by Guy Steele and Gerry Sussman as a medium for teaching new and powerful

by Ernest R. Tello

programming concepts. Today Scheme is considered one of the most modern and progressive of the LISP dialects. Because much of its power is available in a relatively small size, Scheme has many advantages for small machines. This version of the Scheme dialect is one of the relative newcomers in the family of LISP implementations for PCs.

What is so different and so special about Scheme? And how does it compare with Common LISP?

Scheme is like Common LISP in that it supports the lexical scoping of variables, but it also offers several other important and progressive ideas that still have not passed into the LISP mainstream, though it is possible that in the future many of them will.

Although at first it seems as though Scheme is loaded with a kitchen sink of various programming ideas, the underlying basis of it is actually quite simple and well integrated. One important idea in Scheme is that of an environment that can be saved as a context, allowing control to shift temporarily to another such environment and then to return again to the original environment. This may sound like the context switching that is familiar to computer scientists working at the systems level, but here it has a different meaning. Here, the context in terms of an environment means a set of bindings of variables and named functions taken as a unit.

A programming concept that is taken particularly seriously in Scheme is that of first-classness. This idea did not originate with Scheme, but in this dialect of LISP, it has been taken to its most complete expression. Generally speaking, a first-class object is one that has no restrictions on the way it can be used. More specifically, in most programming languages, only numbers, characters, and strings at the most are first-class objects. Even here, frequently only integers are really first class, not numbers in general, and often there may be certain restrictions on the use of one or more of these types of objects in various respects. Usually, for example, you cannot pass arrays, records, and functions as arguments to functions or store them in one another. Even in conventional LISP dialects, some special handling is required when a function is passed as an argument to another function.

In this respect, Scheme is quite radical. The idea is for everything in Scheme to be a first-class object. In PC Scheme, for example, not only procedures but also environments and two other things called continuations and engines can be stored in compound data structures, returned as arguments by a procedure, and bound to variables in three distinct ways.

Functions in LISP are defined through lambda bindings. Because this is strongly analogous to variable binding, some LISP afficionados have wondered why functions are not declared just like variables and lists as:

```
(SETQ [name] lambda [args] function-
                                body)
```

Well, in Scheme, this is exactly what happens, although it is the *SETQ* that is dropped and *DEFINE* is used for the binding of all objects globally with lexical scoping. *SET!* is used only for changing the binding of objects that have already been created. Scheme uses the convention that functions ending in an exclamation point modify their arguments and those that end in a question mark are predicates that return true or false. So, for example, *zerop* in LISP becomes *zero?* in Scheme.

The commitment to making everything in Scheme a first-class object is nothing short of revolutionary. It is staggering to think of the full poten-

tential of a programming system with such capabilities. It is doubtful whether anyone has attempted to take this feature of Scheme to the limits of its power.

This is by no means the only radical concept in the Scheme design, however. Another concept that is important in the Scheme dialect is that of a continuation—a basic concept of control structures in programs. Many of the more familiar LISP control structures such as *catch* and *throw* can be regarded as exemplifying the idea of a continuation, but in Scheme the more general construct is available that enables the more specific ones to be custom-built. Essentially, a continuation is the process to which a computation will progress at a future point in time as has been specified through a programming construct. More specifically, the continuation is the part of a program that can be thought of as waiting for the result of a current computation, and in Scheme such a continuation is a first-class object, just as any current piece of data can be.

Control Structures

Some LISP programmers might be shocked to learn that there are no *PROG*s in Scheme. This is somewhat more of a policy statement than an absolute exclusion, however. Scheme has some other special forms for control that, although they do not specifically replace *PROG*s, certainly do nearly anything that most *PROG* constructs can. One exception to this is *PROG2*, because Scheme does not seem to have a control form that evaluates only the second clause in a sequence. But obviously there are other ways of doing what *PROG2* does. Many stalwart LISP programmers consider the wholesale use of *PROG* constructs to be a crutch to be avoided whenever possible, much like the way structured programming acolytes feel about *GOTO*s.

LETREC is an interesting variant of the *LET* macro that allows a construct called mutually recursive functions, which by definition typically come in pairs. The best way of understanding this is by a specific example. The following one is from the PC Scheme manual; it implements two interdependent functions, *even?* and *odd?*, within the same binding environment, each of which recursively calls the other:

```
(define odd-r-even
  (lambda (n)
    (letrec
      ((even?
         (lambda (n)
           (if (zero? n)
               #!true
               (odd? (-1+ n)))))
       (odd?
         (lambda (n)
           (if (zero? n)
               #!false
               (even? (-1+ n)))))))))
```

The *LETREC* control structure allows two or more lambda procedures to be defined in the same environment. None of the lambda procedures are self-sufficient, but collectively they work in a highly efficient manner by calling one another for parts of their operation.

The Full-Screen Editor

PC Scheme comes with a powerful Emacs-style editor that offers many useful and convenient enhancements, though the speed of many of its operations could be improved upon. One useful innovation is that, as you enter right parentheses, not only does the matching parenthesis become highlighted and blink but also the expression with which the clause begins that is enclosed by these parentheses is printed in a message area down at the bottom of the screen. This is considerably more than a cosmetic enhancement because in those cases in which a long LISP function is used—larger than a single screen—the blinking parenthesis approach alone is useless. This enhancement virtually puts to an end any need for counting parentheses needed at the end of a LISP function on a routine basis, though for debugging, of course, it still never goes away.

Engines

As I indicated briefly earlier, PC Scheme supports an extension that includes a special construct that provides for resource-oriented scheduling. An engine is a special procedure that is given a certain time, measured in ticks that are based on hardware clock interrupts, to complete its computation. It is supervised by two routines called the *a success* procedure and the *a failure* procedure. If the engine's computation is completed before its assigned time has expired, then the success procedure is invoked and the result, together with the number of expired ticks, is returned. If the time expires before the computation is finished, the failure procedure is invoked with the creation of a new engine, which continues the original computation. Although there is no built-in support for allowing a running engine to invoke another engine, the implementation of such nested engine mechanisms is possible using certain special techniques. Engines are particularly well suited for developing discrete time simulation applications, which ordinarily would require multiprocessing support on the operating system level.

SCOOPS

Although object-oriented programming with the message passing paradigm is not a necessary part of the Scheme dialect, it seems to fit in well with the Scheme approach. TI has included a powerful object-oriented extension called SCOOPS with PC Scheme. The two things that make the SCOOPS extension alone more than worth the price of the whole package are its support of multiple inheritance and active values—things that previously were only available on very expensive hardware.

The object-oriented approach used by the SCOOPS package is the nonhierarchical mixins made popular by the Flavor system used on Symbolics LISP machines. Like Smalltalk and most object-oriented programming systems, SCOOPS has a class system and the ability of classes to inherit variables and functions from other classes. But unlike Smalltalk and like Flavors, SCOOPS classes are not limited to inheritance from superclasses in a simple tree hierarchy. Mixins allow classes to be defined that can inherit from any other classes the programmer chooses.

SCOOPS' support for active values—the capability often called procedural attachments in frame-based systems—greatly extends the power of its class system because active values

(continued from page 133)

provide a means to assign a function, or even a complex program, to be evaluated whenever an active value variable is accessed. This provides for association of complex data structures to SCOOPS instance variables and the opportunity for calculated values based on both initial assignments to an instantiated object and to conditions in a dynamically changing environment.

To illustrate the use of active values, you can take a simplified version of the difficult problem of composite objects as used in the CommonLoops system at Xerox PARC. A composite object is an instance of a class that is considered as composed of objects that are instances of other classes, some of which may themselves be composite objects. For example, the body is composed of a head, arms, legs, and torso. The head, hands, and feet can also be represented as composed of other objects such as eyes, ears, fingers, toes, and so on. How can composite objects be implemented in a system such as SCOOPS? Ordinarily, SCOOPS does not even support lists as values of slots or instance variables, much less more complex data structures. At the minimum, a composite object has to contain a list of all of its components. Active values allow this to be done by assigning procedures to instance variables that access external data structures and knowledge structures. The format for specifying an active value is:

```
(instvar
  ([VARIABLE]
    (active [INITIAL-VALUE]
            [GET-FUN]
            [PUT-FUN]
```

Here, *GET-FUN* and *PUT-FUN* represent two procedures—each of one argument only—that are automatically evaluated when the usual *get* and *put* methods for an instance variable are sent to an object. The *INITIAL-VALUE* of the variable is the argument that is passed to these functions.

One way to make the functions access a list—so that when the usual *get* method is used, it returns, for example, a list of the composite object's component parts—is to make the ini-

```
; (C) Copyright 1987 Ernest R. Tello

(define-class composite-object
       (classvars class-part-name class-part-num)
       (instvars (part-names (active parts get-parts add-part))
                 (numbers-of-parts (active '#-parts num-parts
                                                      more-parts)))
     (options
       (gettable-variables class-part-name part-names
                                              numbers-of-parts)
         settable-variables
         inittable-variables))

(define human-body-parts '() )
  (putprop 'human-body-parts 1 'head)
  (putprop 'human-body-parts 1 'neck)
  (putprop 'human-body-parts 2 'arms)
  (putprop 'human-body-parts 2 'hands)
  (putprop 'human-body-parts 1 'trunk)
  (putprop 'human-body-parts 2 'legs)
  (putprop 'human-body-parts 2 'feet)

(define (num-parts p-list)
  (princ p-list))

(define part-map (proplist 'human-body-parts))

(define-class body
     (classvars (class-part-name 'body-parts)(class-part-num
                                          'human-body-parts))
     (mixins composite-object))

(define body-parts '(head neck arms hands trunk legs feet))

(define-method (composite-object put-cpart-name) (new-part)
            (set! body-parts
                 (append (eval (get-class-part-name))
                                   (list new-part))))
(define (add-part new-part)
  (append!  (get-class-part-name) (list new-part)))

(define (get-parts val)
  (princ val))

 (define my-body
  (make-instance body
                'part-names body-parts
                'numbers-of-parts part-map ))

(compile-class composite-object)

(compile-class body)
```

***Example 1:** Composite objects in SCOOPS*

```
[26] (describe body)
    CLASS DESCRIPTION
!
 NAME            : BODY
 CLASS VARS      : (CLASS-PART-NAME CLASS-PART-NUM)
 INSTANCE VARS   : (NUMBERS-OF-PARTS PART-NAMES)
 METHODS          : (GET-CLASS-PART-NAME SET-CLASS-PART-NUM
SET-CLASS-PART-NAME SET-PART-NAMES GET-PART-NAMES SET-NUM-
BERS-OF-PARTS GET-NUMBERS-OF-PARTS PUT-CPART-NAME)
 MIXINS          : (COMPOSITE-OBJECT)
 CLASS COMPILED  : #!TRUE
 CLASS INHERITED : #!TRUE
( )
```

***Example 2:** PC Scheme description of the class* Body *as displayed on the screen*

EVEN MORE POWER AND FLEXIBILITY

BRIEF 2.0

Users and industry press alike have unanimously proclaimed BRIEF as the best program editor available today. Now, the best gets better, with the release of BRIEF 2.0.

Straight from the box, BRIEF offers an exceptional range of features. Many users find that BRIEF is the only editor they'll ever need, with features like real, multi-level Undo, flexible windowing and unlimited file size. But BRIEF has tremendous hidden power in its exclusive macro language. With it, you can turn BRIEF into your own custom editor containing the commands and features you desire. It's fast and easy.

Jerry Pournelle, columnist for BYTE magazine summed it all up by saying BRIEF is, **"Recommended. If you need a general purpose PC programming editor, look no further."** His point of view has been affirmed by rave reviews in C JOURNAL, COMPUTER LANGUAGE, DR. DOBB'S JOURNAL, DATA BASED ADVISOR, INFOWORLD AND PC MAGAZINE.

One user stated "BRIEF is one of the few pieces of software that I would dare call a masterpiece." Order BRIEF now and find out why. **BRIEF 2.0 is just $195. If you already own BRIEF, call for upgrade information.**

TO ORDER CALL: 1-800-821-2492 (in MA call 617-659-1571)

As always, BRIEF comes with a 30 day money-back satisfaction guarantee.

Solution Systems™

335 Washington St.
Norwell, MA 02061
(617) 659-1571

Look at these BRIEF 2.0 enhancements!

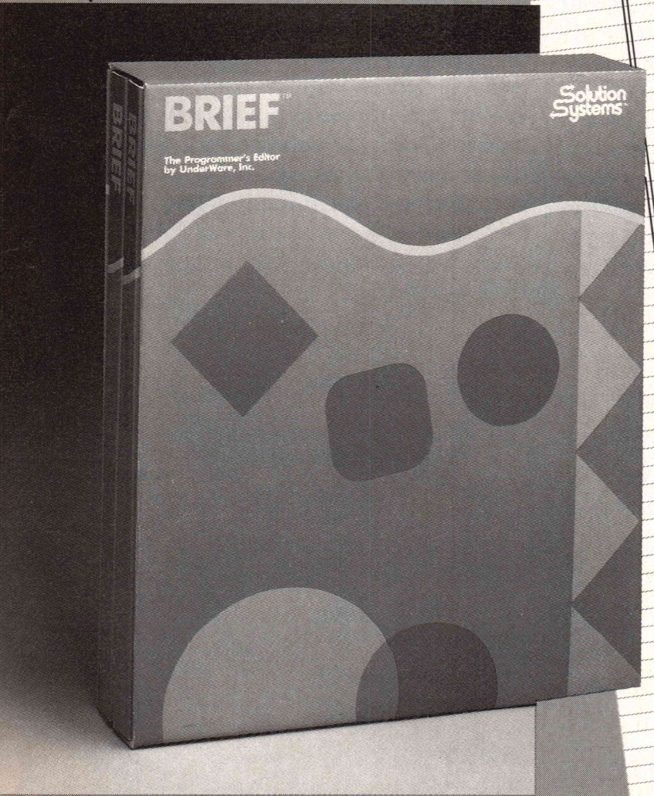
Main Features:

- All new documentation with tutorials on basic editing, regular expressions **and** the BRIEF Macro Language.
- Setup program for easy installation and configuration. (Requires no knowledge of the macro language)
- Increased speed for sophisticated operations like Undo and Regular Expression Search.
- Expanded regular expressions, with matching over line boundaries.
- More block types, with marking by character, line or column.
- Command line editing (move cursor, add and delete characters, specify command parameters).
- Support for more programming languages.
- Optional borderless windows.
- Enhanced large display support, including wider displays.
- Reconfigurable indenting for C files (supports most indenting styles).

Plus the basic features that made BRIEF **SO** popular!

Basic Features:

- Full multi-level Undo
- Windows
- Edit many files at once
- File size limited only by disk space
- Automatic language sensitive indentation



Requires an IBM PC or compatible with at least 192K RAM.
BRIEF is a trademark of UnderWare, Inc.
Solution Systems is a trademark of Solution Systems.

CIRCLE 142 ON READER SERVICE CARD

(continued from page 134)

tial value of the active value variable the name of the list of parts and define the *get* and *put* functions such that they can return and append this list. The appending function is tricky to implement because it needs the name of the list so that things can be appended to it. Because it is desirable to make such a function as general as possible so that it can be used with any composite object, it has to have a way of finding the name of the specific list of parts that applies only to the particular object in question. The problem is that the initial value, which is the name of the list in question, is not returned by its normal *get* function anymore but is passed as an argument to the active value function. One way to get around this problem is to use another variable that is not an active value variable to store the name of the list where it may be easily accessed by a global function.

I'll now illustrate a successful application of this strategy in PC Scheme. Two classes have been implemented—*Composite-Object* and *Body*—where the first is a mixin, or superclass, for the second. The code for these classes is supplied in Example 1, page 134. When you ask Scheme to describe the class *Body,* you see the screen display shown in Example 2, page 134. The variables of *Body* have all been inherited from *Composite-Object*. The variable *part-names* is implemented as an active value that accesses a list and prints its contents when called. The variable *numbers-of-parts* is also an active value, but in this case, its *get* function returns and prints a property list that contains a list of body parts each with the property of how many such parts the body should contain. One possible extension of this example would be to define various subclasses for different types of organisms. So, for example, humans, horses, ants, spiders, and centipedes would have different entries on their property lists for the number of legs.

Using the current example I created an instance of the *Body* class called *My-Body*. I set the values of its variables so as to reference appropriate lists and property lists for the names and numbers of its parts. Then, sending it the messages indicated produced the results:

```
[27] (send my-body get-part-names)
(HEAD NECK ARMS HANDS TRUNK LEGS
                            FEET)

[28] (send my-body get-numbers-of-
                            parts)
(FEET 2 LEGS 2 TRUNK 1 HANDS 2 ARMS 2
                     NECK 1 HEAD 1)
```

One of the nice discoveries I made when developing this use of active values is that, once this interface to more complex auxiliary data structures has been correctly implemented, the values will then actually be displayed as if they are part of the object when the *describe* function is called. So, in the case of the object *My-Body*, which is an instance of the *Body* class, the following result is returned when requesting its description:

```
[30] (describe my-body)
       INSTANCE DESCRIPTION
!
Instance of Class BODY
 Class Variables :
 CLASS-PART-NAME : BODY-PARTS
 CLASS-PART-NUM : HUMAN-BODY-
                       PARTS

 Instance Variables :
 NUMBERS-OF-PARTS : (FEET 2 LEGS 2
 TRUNK 1 HANDS 2 ARMS 2 NECK 1 HEAD 1)
```

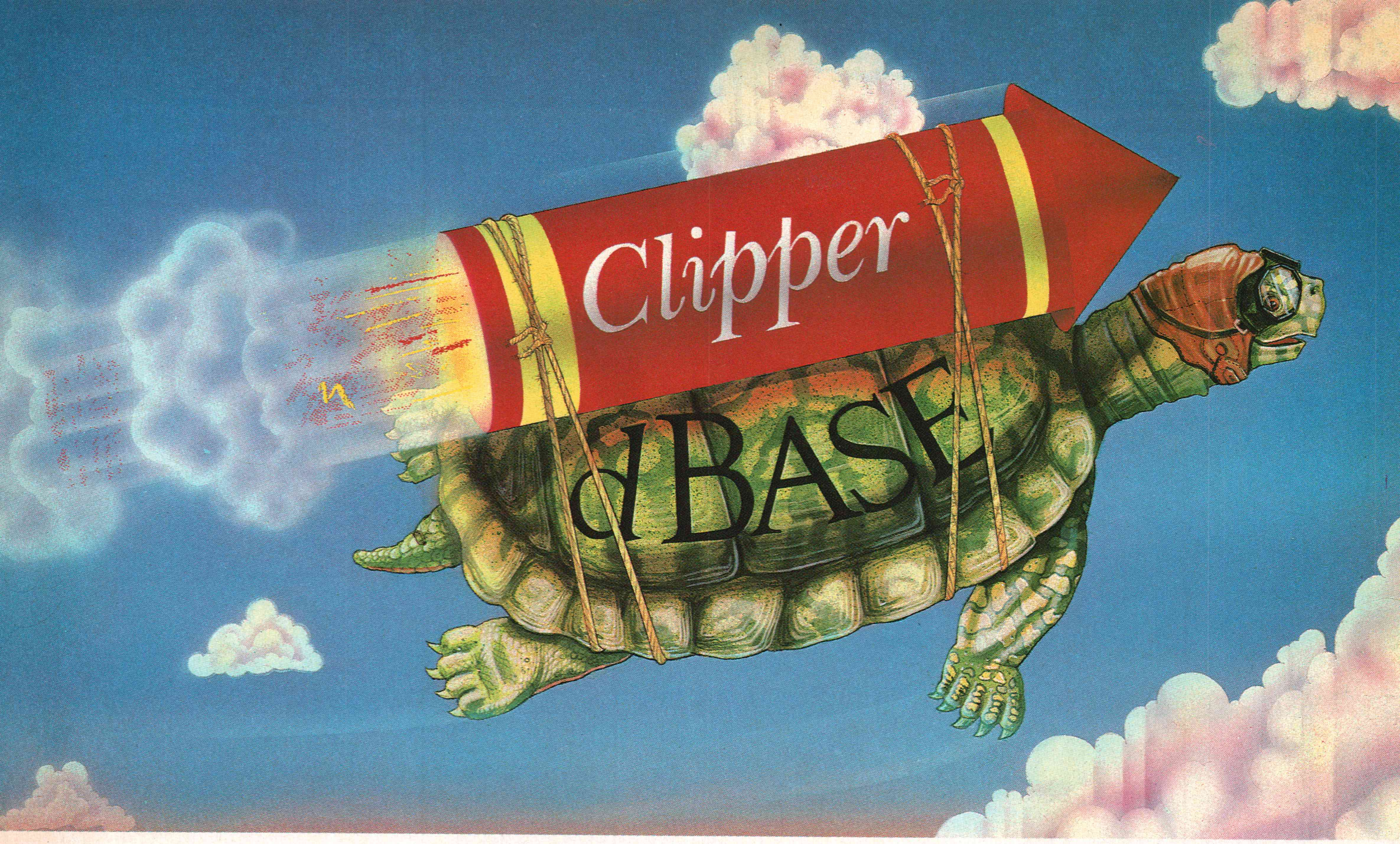
In the full implementation of the *Composite-Object* class, there would be various additional methods, including one that could automatically initialize the objects that were part of any instance of this class or any of those of which it was a mixin. This method would include a recursive procedure that would access the *part-names* slot to get a list of all of its components and then the *numbers-of-parts* property list to find out how many of each were needed. You could then repeat this for all of those parts that were also composite objects, until all the objects being instantiated were simple and not composite. As a variant, you could build a subclass of *Composite-Object*, like the *Perspective* class in KRL and Loops, in which the objects that were its components would only be instantiated on request. The idea of a *Perspective* is a composite whose components are not exactly parts but various different roles in which the individual object participates. In this sense, it would be as if the individual had various distinct aspects, each of which could independently be members of different classes.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 8.**

Vendor

PC Scheme
Texas Instruments
12501 Research Blvd., MS 2223
Austin, TX 78769
(512) 250-7533
Reader Service No. 43



Turtle Souped

Real programmers don't use dBASE. Or do they?

We're finding that some very swift programmers are using it to write some very fast applications, and are completing their projects much more quickly.

But they cheat.

They use our Clipper™ compiler to combine dBASE™ with C and assembler.

With dBASE used like pseudo-code, they can then quickly create prototypes that actually run.

Then, with dBASE doing the high-level database functions, they use our Clipper compiler to link in C or assembly language modules from their own bag of tricks.

And they're finding that they're linking in less than they expected because Clipper compiled code runs so fast and because of Clipper's built-in enhancements.

Clipper includes easy networking that provides file and record locking the way it should be done.

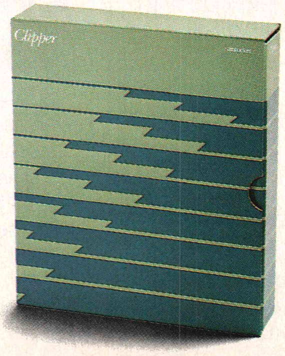
Fast screens that can be treated as memory variables and eliminate the need for direct screen writes and all that tortuous heap management code.

Box commands that make windowing a breeze. And more.

So if you'd like to use your time more productively, check us out: Nantucket Corporation, 12555 W. Jefferson Boulevard, Los Angeles, CA 90066.

Or if you're on deadline, call (213) 390-7923 today.

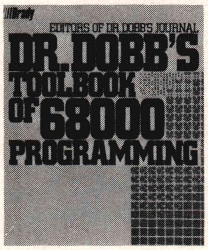
Clipper could get you out of the soup.



Nantucket®

Dr. Dobb's Toolbook Shelf

Dr. Dobb's Toolbook of 68000 Programming



In this complete collection of practical programming tips and techniques for the 68000 family, you'll find the best articles on 68000 programming ever published in Dr. Dobb's, along with new material from 68000 experts. You'll find a set of development tools, routines, useful applications, and examples to show you why computers using the 68000 chip family are easy to design, produce and upgrade. All programs are available on disk.

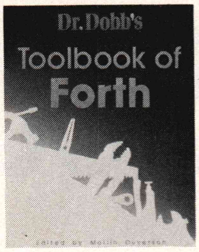
Please specify one of the following disk formats: MS-DOS, CP/M 8", Osborne, Macintosh, Amiga, Atari 520st.

68000 Cross Assembler

An executable version of the 68000 Cross-Assembler discussed in the book is also available, complete with source code and documentation. Requires CP/M 2.2 with 64K or MS-DOS with 128K. Specify: 8" SS/SD, Osborne, MS-DOS

68000 Toolbook	**Item #040 $29.95**
68000 Toolbook with disk	**Item #041 $49.95**
68000 Cross Assembler	**Item #042 $25.00**

Dr. Dobb's Toolbook of Forth

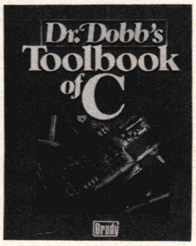


This comprehensive collection of useful Forth programs and tutorials contains Dr. Dobb's best Forth articles, expanded and revised, along with new material. Contents include: Mathematics in Forth, Modifications/ Extensions, Forth Programs, Forth--The Language, Implementing Forth. All screens in the book are also available on disk as ASCII files.

Toolbook of Forth #030 $22.95
Toolbook of Forth with disk Item #031 $39.95

Please specify one of the following disk formats: MS/PC-DOS, Apple II, Macintosh, CP/M. For CP/M disks, specify Osborne or 8" SS/SD.

Dr. Dobb's Toolbook of C



This authoritative reference contains over 700 pages, including the best C articles from Dr. Dobb's Journal, along with new material. You'll find hundreds of pages of valuable C source code, including a complete compiler, an assembler, and text processing programs.

Toolbook of C Item #005 $29.95

VIEWPOINT
(continued from page 14)

Finally, print out a copy of every program that does not print out a copy of itself.

This program, in the process of generating all programs, will eventually generate itself. Does it print out a copy of itself? If it does, it is breaking the rule by printing out a copy of a program that prints out a copy of itself. If it does not, it is breaking the rule by failing to print out a copy of a program that does not print out a copy of itself. This fatal contradiction proves that the halting problem has no solution.

You may recognize this as Russell's paradox (the set of all sets that do not contain themselves) or as the barber paradox (the barber who shaves every man who does not shave himself).

Any problem that a debugger can convert to the halting problem, such as the string-output problem, is equally unsolvable. Some other obvious examples are:

1. determining whether a program will reach a specified point (Ada programmers: this is why *PROGRAM_ERROR* has to be a run-time error, not a compile-time error)
2. determining whether a variable is initialized before it is used
3. determining whether a given segment of code is inaccessible and will never be executed
4. determining whether two programs do the same thing

Of course, a debugger or compiler can sometimes predict such errors—for example, inaccessible code can sometimes be identified at compile time. But universal solutions to such problems do not exist.

The impossibility of determining whether two programs do the same thing means that it is always possible to defeat a certain kind of Trojan horse. In a lecture reprinted in the *Notices of the ACM* (August 1984), Ken Thompson argued that he could put a Trojan horse into a C compiler that would miscompile the *login* statement to allow him access to any Unix system compiled with it, and it would miscompile the C compiler to insert a copy of itself. The Trojan horse itself would not appear in the source code for the C compiler. In a letter to the editor, Steve Draper noted that such a Trojan horse can be defeated by paraphrasing the C compiler (writing different code that does the same thing) and then recompiling it. No Trojan horse can infallibly recognize paraphrased programs—hence there is always a paraphrase that will defeat the Trojan horse.

My own opinion in this matter is that, unless the Trojan horse were skillfully written, most paraphrases would defeat it, and in fact it would probably be defeated eventually by normal software maintenance. Any Trojan horse smart enough to recognize most paraphrases would probably be much larger than the rest of the C compiler. You'd never get it through the gates.

The halting problem is intimately related to two other problems, which were posed by the mathematician David Hilbert in 1900. Is there a formal proof or disproof for every mathematical statement? Is there an algorithm to find proofs?

The first question was answered in the negative by Kurt Goedel in 1931. Goedel's proof was complex, but if you accept the unsolvability of the halting problem, it can be proved simply. Whether a particular program halts is a mathematical statement. In fact, many mathematical theorems are already special cases of the halting problem because you can write a program to search for counterexamples and halt when it finds one. The theorem is equivalent to the assertion that the program never halts.

If there were always a formal proof or disproof of the assertion that a program halts, then you could simply generate all proofs (more or less as the program described earlier generated all programs) until you found either a proof or a disproof. That would solve the halting problem. Because the halting problem is in general unsolvable, there must be at least one mathematical statement of this kind that is undecidable—that is, it cannot be formally proved or disproved.

This shows that it is impossible in general to prove that a program works. Specific programs or limited classes of programs can be proved to do certain things, but there is no way

to do this for every program.

Given that some mathematical statements are undecidable, is there a program, the "decidability program," that can tell whether any mathematical statement is decidable, even without deciding whether it is true or false? As you might have guessed from the tone of this article, the answer is again no. If you have a decidability program, you can take any program and ask whether it halts. Then apply the decidability program to this question. If the question is decidable, a search of all proofs will prove it or disprove it. If the question is undecidable, then the program never halts; otherwise, you could prove that it halts by simply running it until it halts.

Therefore, theorem-proving programs, however successful they might be in limited areas, can never prove everything. Some things must always remain beyond their grasp.

These arguments are not rigorous in the mathematical sense because too much has been left out. A major part of Turing's and Goedel's work involved formalization of the concepts of "computation" and "proof" to the point at which their arguments would be accepted by mathematicians.

You may have already spotted one tacit assumption that does not correspond to reality. The programs are not constrained by memory limitations. If a program does have a memory limitation, then the halting problem can in theory be solved—but only by a program with a much larger memory.

This is how it can be done. A program with a memory limitation has only a finite number of states. A debugger can single-step it, keeping track of the states it has occupied. If it occupies the same state twice before halting, it will repeat the same sequence of states indefinitely and will never stop.

To do this, the debugger needs enough memory to keep track of which states the program has occupied. Only one bit is required for each possible state, but the number of possible states for even a simple program is truly mind-boggling. Every combination of bits in the memory is a different state. Hence a program with only 1,024 bytes of memory has at least $2^{(1024 \times 8)}$ states due to memory configuration alone, to say nothing of flags and registers. This number of flip-flops would not fit into the entire known universe. It can therefore be said that the halting problem has no solution even in this case.

It should be clear, then, that there are definitely some limits to what artificial intelligence can accomplish and that mathematicians' and programmers' jobs can never be completely automated. (This is a great comfort to me because I am a mathematician and programmer.)

Only perfect solutions are impossible, however. It can still be argued, and it is argued by some, that artificial intelligence programs will eventually be able to solve every problem that the human mind can solve, with at least the same success rate. And if the only requirement is practical solutions, not perfect solutions, then many interesting but theoretically unsolvable problems can be solved.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 1.**

Can You Afford A Million $ Mistake?

The purchase of your business computer could cost you thousands, maybe millions of dollars.

For just $19.97 *BUSINESS SOFTWARE* can help you get what you paid for and more!

BUSINESS SOFTWARE is for the experienced PC user who wants to squeeze the utmost potential from existing equipment AND get full value from new software products being considered.

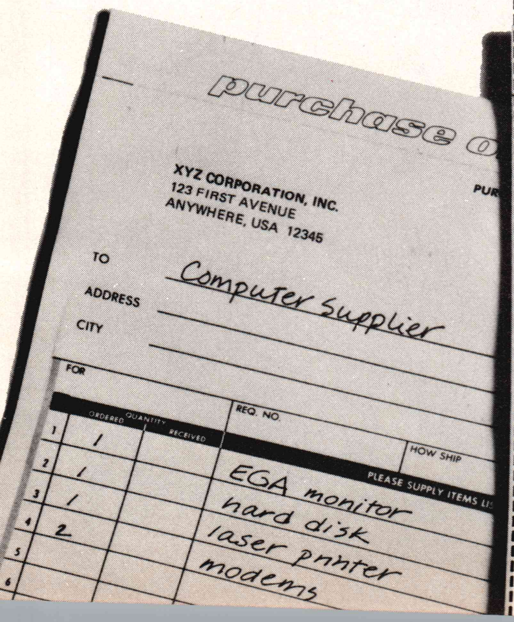
It only makes "cents" to get a fair return on your investment.

- Fast-paced tutorials covering the advanced features of popular software,
- Valuable tips, traps and templates,
- Benchmark style reviews and comprehensive comparisons,
- Advanced macros and actual program examples,
- In-depth software evaluations by eminent experts.

BUSINESS SOFTWARE delivers editorial that is Read . . . Understood . . . and Put to Work.

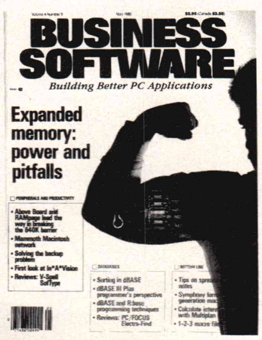
Why not start your subscription to *BUSINESS SOFTWARE* today—you'll not only save over 40% off the newsstand price, but enjoy the convenience of having *BUSINESS SOFTWARE* delivered right to your doorstep each month.

Take advantage of this low subscription price today!



BUSINESS SOFTWARE

40% SAVINGS



Yes! Enroll my subscription to *BUSINESS SOFTWARE* at a savings of more than 40% off the newstand price

☐ 1 Year (12 issues) for **$19.97** —a 40% discount!
☐ 2 Years (24 issues) for **$35.97** —a 50% discount!

☐ Payment enclosed ☐ Bill me later

Name ______________________
Company ______________________
Address ______________________
City ____________ State ________ Zip ______

Mail to: Business Software Magazine, P.O. Box 27975, San Diego, CA 92128.

Savings based on full one-year newsstand price of $35.40. In Canada add $7 for surface mail per year; other countries add $15 for surface mail per year, $37 for airmail per year. All foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

HOT GRAPHICS PACKAGE FOR C PROGRAMS.* $39.95

Everything you need to write dramatic graphics effects into your Eco-C88 C programs. Some of the features include:

- Support for EGA, CGA, and Z100
- Over 100 graphics and support functions, many of which are PLOT-10 compatible.
- Many low level support routines reside outside your small model code-data area
- Can write dots thru the BIOS (for compatibility) or to memory (for speed)
- Graphics function help from CED editor available
- World, pixel or turtle color graphics modes
- 47 standard fill patterns, 17 line dashing patterns, Hershey fonts, plus user defineable fill, dash and fonts
- Supports view areas, rotateable fonts, clipping, arbitrary fill areas, extensive error checking, examples, and user's manual.

A must for the graphics enthusiast and a bargain at only

$39.95

***Requires Eco-C88 C Compiler.**

NEW POP-UP WINDOWS FOR YOUR C PROGRAMS.

This windowing library allows you to add pop up windows in your C programs quickly and easily. Use them for help windows, selection menus, error messages, special effects—anywhere you need an attention getter. Just some of the features include:

- CGA, EGA, and monochrome support
- Slow mode option for "flicker" displays
- Control any program that goes through the BIOS
- Use up to 255 windows
- No special window commands; use print f ()
- Resize and move windows
- Custom window titles and borders
- Can be used with ANSI device driver
- Most of window's code-data lies outside small model limits
- Use any of the IBM text or block characters
- User's manual and examples

The Windowing Library requires an IBM PC compatible BIOS and the Eco-C88 C compiler.

ONLY $29.95

HANDY LIBRARIAN MAKES LIFE EASIER.

Now you can combine your modules, functions, and subroutines into your own library for easy link commands. Fully compatible with ANY standard OBJ format files (not just Ecosoft's products). With the Ecosoft librarian, you can:

- Add, delete, and extract from a library
- Get table of contents or index of a library
- Combine libraries, control library page size, use switches for combinations, process complex library requests, use wildcards, and do library directives from command files.
- Complete with user's manual

A valuable addition for any programmer.

ONLY $29.95

THE FIRST PROFESSIONAL 'C' COMPILER FOR UNDER $60.

A C compiler with many ANSI enhancements at an unbelievably low price. The Eco-C88 C compiler has:

- Prototyping (the new type-checking enhancement)
- Enum and void data types
- Structure passing and assignment
- All operators and data types
- A standard library with more than 200 functions (many of which are System V compatible for greater code portability)
- CC and mini-make that all but automates the compile process
- 8087 support (we sense the 8087 at runtime — no dual libraries)
- ASM or OBJ output for use with MSDOS linker
- Tiered error messages — enable-disable lint-like error checking
- Fast compiles and executing code
- Expanded user's manual
- CED full-screen program editor

Everything you need at the unbelievable price of $59.95.

Eco-C88 C compiler requires an IBM PC, XT, or AT (or compatible) with 256K of memory, 2 disk drives and MSDOS 2.1 or later.

Orders only:
1-800-952-0472
Technical Information:
(317) 255-6476

NOT COPY PROTECTED.

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

ORDER FORM CLIP & MAIL TO: Ecosoft Inc., 6413 N. College Ave., Indianapolis, IN 46220

ITEM	PRICE	QTY	TOTAL
Flexi-Graph Graphics	$39.95		
Window Library	$29.95		
Eco-Lib Librarian	$29.95		
Eco-C88 C Compiler CED	$59.95		
	SHIPPING		
	TOTAL (IND. RES. ADD 5% TAX)		

PAYMENT: ☐ **VISA** ☐ **MC** ☐ **AE** ☐ **CHECK**

CARD # ______ EXPIR DATE ______

NAME ______

ADDRESS ______

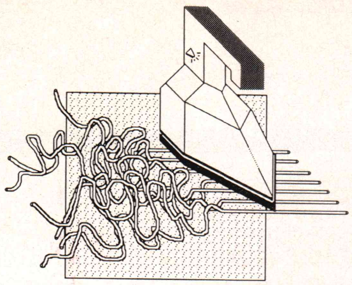
CITY ______ STATE ______

ZIP ______ PHONE ______

CIRCLE 89 ON READER SERVICE CARD

ECOSOFT

THE STATE OF BASIC



New BASIC Subroutines Continued

In this issue, we resume the discussion about subroutines and look at how they are implemented in Summit Software's BetterBASIC and Borland's new Turbo BASIC.

Subroutines in BetterBASIC strongly resemble Pascal procedures. As a matter of fact, the *PROCEDURE: <name>* syntax is used to declare a BetterBASIC subroutine. Once you type *PROCEDURE:* followed by a subroutine name, BetterBASIC creates a new workspace for that subroutine. This gives subroutines a great degree of freedom because they can have their own line numbers (1 to 32,767) and local variables. Unlike Pascal procedure declarations, subroutine argument lists in BetterBASIC are declared on separate lines.

Example 1, below, shows a simple subroutine, *Increase*, to increment an integer. Notice that *PROCEDURE: Increase* is typed on one line and its arguments, variables *I* and *J*, are on following separate lines. BetterBASIC requires the use of the keyword *ARG* to distinguish between an argument and a local variable. The */VAR* is used to declare that a variable is passed by reference; otherwise, as in the case of *J*, it is passed by value. The /OPT is a directive that assigns a default value to a variable.

Example 2, below, shows the simple main program that calls the procedure *Increase* twice. In the first call, *Increase* is supplied with one argument (corresponding to variable *I* in the declaration part of the procedure *Increase*). This causes the default value of 1 to be assigned to variable *J*. In the second call to *Increase*, two arguments are used, causing the value of 2 to be passed to argument *J*.

The BetterBASIC feature of assigning default values to variables "omitted" while calling a subroutine has a parallel feature in the Ada language. To avoid chaos in subroutine calls when using this feature, you must observe the following rule: you cannot "skip" arguments. In other words, once you rely on the default value of an argument, all the arguments that follow must have default values, which must be invoked. You cannot pick and choose. To use the default-value feature in a BetterBASIC subroutine, divide your argument list into two logical sets of parameters. The first set should always require values to be exchanged with the subroutine; hence, the parameters must be present during a subroutine call. The second set consists of parameters that should have logically related default values (we say *logically* to stress that these default values are all attributes to a single default state). As a result, these parameters are either present (to provide data for a nondefault state) or absent (to refer to the default-state value). Another approach to using the second parameter set can be related to the fact that its default values cannot be attributes to a finite default state. In this case, arrange the declaration of the subroutine parameters in a sorted order based on the overall probability of not using a default value. This places

```
PROCEDURE: Increase
INTEGER ARG: I/VAR
INTEGER ARG: J/OPT=1
10 I = I + J
END PROCEDURE
```

***Example 1:** BetterBASIC procedure to increment an integer variable*

```
INTEGER A, B
10 A = 10
20 B = 12
30 Increase A ' increment with default value of 1
40 PRINT A ' prints 11 = 10 + 1 (default)
50 Increase A, B
60 PRINT A ' prints 13 = 11 + 2
70 END
```

***Example 2:** BetterBASIC demonstration program using procedure* Increase

```
SUB Stat(X#(2), Col%, Average#, Sdev#) STATIC

LOCAL Sum#, SumX#, SumXX#, Row%

Sum#   = 0.0
SumX#  = 0.0
SumXX# = 0.0

FOR Row% = LBound (X#,1) TO UBound (X#,1)
 Sum#   = Sum#   + 1.0
 SumX#  = SumX#  + X#(Row%, Col%)
 SumXX# = SumXX# = X#(Row%, Col%) ^2
NEXT Row%

IF Sum# ) 1.0 THEN
 Average# = SumX# / Sum#
 Sdev# = SQR((SumXX# - SumX# ^2 / Sum#) /
                                        (Sum# - 1.0)
ELSE ' code for insufficient data
 Average# = -1.0E+30
 Sdev# = -1.0E+30
END IF

END SUB
```

***Example 3:** Turbo BASIC subroutine to obtain the average and standard deviation of data stored in an array*

parameters that more seldomly resort to default values in the beginning of the list and vice versa.

Looking at Example 2, you may observe more differences in syntax between BetterBASIC and the other BASIC implementations we discussed in the previous column—for example, the *CALL* keyword and parentheses are not used in BetterBASIC.

Because BetterBASIC supports Pascal-like record structures, you can use them to pass many variables and still keep a short, formal argument list. This feature enables BetterBASIC to refrain from supporting *SHARED* variables (as in QuickBASIC) to keep the argument list small. Passing record-type arguments in BetterBASIC subroutines is even safer because you maintain tighter control over shared data and greatly minimize any undesirable side effects.

BetterBASIC does not offer built-in functions to return the lower and upper bounds of arrays, which makes writing general-purpose, array-manipulating routines a bit more involved. Because the lower bound of any array can be 0 or 1, you can write such routines to start at index 1. Your routines must rely heavily on integer-type parameters that supply the upper array bounds, however. Such reliance makes the routines extremely vulnerable to corrupted upper bounds values; there is no easy way to compare these parameters with the actual array bounds they represent. The positive side of using such parameters is that, when arrays are not fully populated with valid data, you still need to supply data counters. Thus, the upper array bound parameters frequently end up being used as data counters.

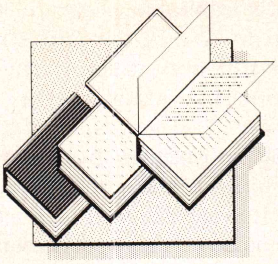
Turbo BASIC implements subroutines in a manner that resembles QuickBASIC: *GOSUB* using labels to direct the jumps to subroutines and named subroutines. A callable subroutine (or procedure as it is called in Turbo BASIC) can have an argument list to pass arguments that are scalar and/or array variables. With Turbo BASIC you must specify the number of dimensions of an array using an integer constant enclosed in parentheses following the array name. Like QuickBASIC and True BASIC, variables are passed by reference and expressions are passed by value. Turbo BASIC supports *STATIC* and *RECURSIVE* subroutines as well as *LOCAL*, *STATIC*, and *SHARED* attributes for variables in a subroutine. The *LOCAL* attribute declares a variable to have a scope and visibility limited to the subroutine. Local arrays must be dimensioned as dynamic arrays (Turbo BASIC supports static and dynamic array dimensioning). Static variables retain their values between subroutine calls, whereas shared variables become global to the rest of the program. Turbo BASIC also implements *EXIT SUB* to enable program flow to return to the caller and supplies built-in functions to return the lower and upper bounds of an array.

Example 3, page 144, shows a Turbo BASIC subroutine that returns the basic statistics and data stored in an array. It is similar to the QuickBASIC and True BASIC versions presented in the last column.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 9**.

BOOKS



Numerical Recipes: The Art of Scientific Computing. Press, William H.; Flannery, Brian P.; Teukolsky, Saul A.; and Vetterling, William T. Cambridge, England: Cambridge University Press, 1986. $40.

In the beginning, there was Hamming (1962). And Hamming begat Acton (1970), who begat Dahlquist and Bjorck (1974), who begat Ralston and Rabinowitz (1978), who begat Stoer and Burlisch (1980). Now we have *Numerical Recipes*, the latest addition to a fine line. For those who have worked with numerical analysis, it is a welcome addition to your arsenal; for those of you who are new to the subject or who have not yet begun to build your library, it is the one book to buy if you are going to have to solve anything numerically on a computer.

Content first—the book is comprehensive. It has the usual chapters on the solution of linear algebraic equations, interpolation and extrapolation, integration of functions, evaluation of functions, root finding and nonlinear sets of equations, minimization or maximization of functions, eigensystems, integration of ordinary differential equations, two-point boundary value problems, and an introduction to partial differential equations. In addition, it contains a collection of chapters on topics not usually found in other books: special functions, random numbers, sorting, Fourier transform spectral methods, statistical description of data, and modeling of data. These chapters, together with the others, group in one place almost all the techniques that today's scientists and engineers commonly use to get the job done.

Issues of style come next. The knowledge of mathematics required to cope with the text is university level—that is, you should have some familiarity with linear algebra and calculus. The authors stay away from what I would consider an overly theoretical approach in both the text and the mathematical notation, although you do have to understand the normal amount of symbolic manipulation that comes when dealing with matrices, sums, and integrals.

In each subject area, the authors present several methods after discussing the problem in the introductory section of the chapter. Each section covers a method, with text and some equations as appropriate. Usually there is a FORTRAN subroutine at the end of the section, which not only illustrates the method but which can also be used to get real work done. (The authors have also translated all the subroutines—there are more than 200 of them—into Pascal and have included these in an appendix.) As they discuss each alternate method, the authors give you their candid opinion of the strengths and weaknesses of the approach, usually placing it in some historical context. There are references at the end of each section and a great bibliography at the end of the book. The references and bibliography alone are an invaluable source for getting more information when you need to go further.

The book's layout is clean. The type is easy to read, the choice of notation is excellent, and the programs are easy to follow because they are well commented. It is a little unfortunate that FORTRAN was chosen as the in-text language; this reflects FORTRAN's omnipresence in scientific computing but carries with it the terrible burden of short, elided names for variables. The Pascal versions suffer from being machine-translated from the FORTRAN. The authors promise to swap the FORTRAN and Pascal roles in the next edition and would like to hear from those folks who would like to see a C version. My vote would be for a clean implementation in Ada.

The authors have style. They chose the name of the book to be reminiscent of a cookbook, but in their words, there is a difference between a cookbook and a restaurant menu: "The latter presents choices among complete dishes in each of which the individual flavors are blended and disguised. The former—and this book—reveals the individual ingredients and explains how they are prepared and combined." The strength of the book is that with each recipe (read: computer subroutine), there is enough explanatory underpinning so that, with a reasonable amount of care and intelligence on the part of the reader, the proverbial bullet in the foot can be avoided. The authors' writing style makes the material easy to follow and not dull reading at all; it's great to see hard-earned experience come through as charm and wit.

Although this book owes a debt to all its predecessors—most notably for its acknowledged stylistic similarity to Acton's—it is different in that it takes positions and makes judgments. It is a guidebook, where others are compendia. It consciously does not spend a lot of time on methods that the authors feel have been popular in the past but have now, perhaps recently, been superseded by others.

Supplementary materials, which I have not seen, are available—machine readable source for the subroutines in either language, example books that show how to use the subroutines, and machine readable versions of the example books. To get one complete set (one language) of everything costs less than $60, which when coupled with the price of the book, is less than $100. When you consider the amount of time it would take you to get a routine working when you need one, you would have to have an extremely low hourly rate not to be able to easily justify the cost of these materials.

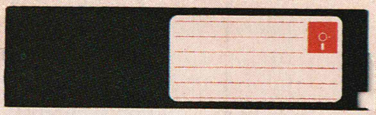
—Joe Marasco

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 10.**

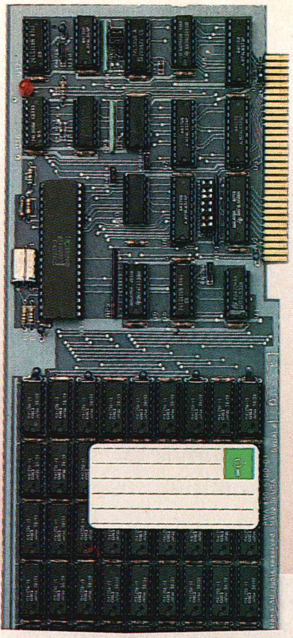
Think fast! Pick the better fit...

Our 5th Year Bonus!
Mention this ad when ordering
and get your choice of a V 20-8
(replaces 8088) or $20 off on
your Battery Backup purchase!



FLOPPY DISK.

- Fills time between coffee breaks
- Makes a hard disk seem *fast*
- Your computer appears busy (even if you aren't!)
- Wears out moving parts



SEMIDISK Disk Emulator.

- Gets that job done NOW
- Makes a hard disk seem *slow*
- Maximizes your productivity with anything from databases to compilers
- Totally silent operation

...for YOUR demanding tasks.

SURPRISE! *Neither* is memory mapped, so they don't affect your precious Main Memory. *Both* retain data indefinitely - even with the computer turned off.

THE SEMIDISK SOLUTION. You could invest in a series of "upgrades" that turn out to be expensive band-aids without solving your real problem. Even those "Accelerator" and "Turbo" boards do little to speed up disk-bound computers. If your applications spend too much time reading and writing to disk (and whose don't?), you won't want to settle for anything less than a SemiDisk disk emulator. The SemiDisk comes in 512K and 2Mb capacity. More boards may be added to make up to an 8 Megabyte SemiDrive!

SPEED THAT'S COMPATIBLE. PC, XT or AT, if you need speed, the SemiDisk has it. How fast? Recent benchmarks show the SemiDisk is from 2 to 5 times faster than hard disks, and from 25% faster (writing) to several times faster (random reads) than VDISK and other RAMdisk software that gobble up your main memory.

MEMORY THAT'S STORAGE. Using our small external power supply, with battery backup, your data remains intact through your longest vacation or even a seven-hour power failure!

CELEBRATE WITH US! Now, SemiDisk celebrates its fifth birthday with a special offer for IBM-PC owners. Buy a SemiDisk now and we'll include an 8 MHz V-20 microprocessor (replaces the 8088) to make your new SemiDisk run even faster. Don't need the V-20? We'll take $20 off the price of your Battery Backup Unit!

	512K	2Mbyte
IBM, PC, XT, AT	**$495**	**$ 795**
Epson QX-10	**$495**	**$ 995**
S-100 SemiDisk II	**$795**	**$1295**
S-100 SemiDisk I	**$299**	-------------
TRS-80 II, 12, 16	**$495**	**$ 995**
Battery Backup	**$130**	**$ 130**

Someday you'll get a SemiDisk.
Until then, you'll just have to...wait.

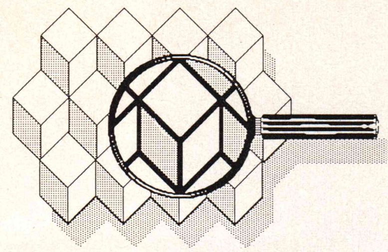
SemiDisk

MasterCard VISA

SemiDisk Systems, Inc., 11080 S.W. Allen Blvd., Beaverton, Oregon 97005 (503) 626-3104

CIRCLE 85 ON READER SERVICE CARD

OF INTEREST



For the Amiga

Metacomco, the author of AmigaDOS, has released an improved version of the Amiga command-line interpreter called Shell. Shell is a programming environment featuring command-line history and editing, aliases, path and push/pop directories, and variables. Shell is compatible with all standard command-line interpreters and sells for $69.95. Reader Service No. 16.
Metecomco
5353 Scotts Valley Dr., #E
Scotts Valley, CA 95066
(408) 438-7201

The latest version of the **Lattice** AmigaDOS C compiler has a new library with more than 100 functions and increased library modularity. Version 3.10 also features new addressing modes, fast pointer and integer math, fast IEEE floating-point routines, and multitasking support. The base-level compiler sells for $225; the professional package (which includes text management and make utilities, screen editor, and debugger) costs $375. Reader Service No. 17.
Lattice Inc.
P.O. Box 3072
Glen Ellyn, IL 60138
(312) 858-7950

Central Coast Software has enhanced DOS-2-DOS, a disk-to-disk file transfer program for the Amiga that transfers all MS-DOS file types to and from AmigaDOS. DOS-2-DOS now supports 3½-inch 720K disks, formats both 3½- and 5¼-inch MS-DOS disks, converts ASCII file line-ending characters, and provides WordStar compatibility. The program costs $55. Reader Service No. 18.
Central Coast Software
268 Bowie Dr.
Los Osos, CA 93402
(805) 528-4906

For the Macintosh

The Complete Book of Macintosh Assembly Language Programming, Volume II, by Dan Weston features a collection of assembly-language projects that explore advanced topics in Mac programming. Published by **Scott, Foresman & Co.**, the book covers the new ROM of the Mac Plus, how the clipboard is used and how it is converted by Switcher, and how to use the hierarchical file system and includes source code listings. It costs $22.95. Reader Service No. 19.
Scott, Foresman & Co.
1900 E. Lake Ave.
Glenview, IL 60025
(312) 729-3000

The APL*PLUS System for the Macintosh from **STSC** is a full-featured APL language interpreter. The system is compatible with STSC's APL*PLUS System for the IBM PC and allows existing applications to be converted and run on the Mac. The package takes advantage of standard Mac features, and common desk accessories can be used from the APL environment. The APL*PLUS System for the Macintosh runs on a Macintosh with at least 512K RAM and one disk drive and sells for $395. Reader Service No. 20.
STSC Inc.
2115 E. Jefferson St.
Rockville, MD 20852
(800) 592-0050

Miscellaneous

Real-Time Computer Science Corp. (RTCS) is now shipping RTX286, a real-time, multitasking, multiuser operating system for the IBM PC/AT. RTX286 is a complete implementation of Intel's iRMX286 operating system specifically configured for the AT and its peripheral devices. It takes advantage of the protected mode of the iAPX286 processor, offering memory-access protection as well as allowing users to access up to 16 megabytes of memory directly. RTX286 is priced at $2,395, and RTX286-C (a configurable version) costs $2,795. Reader Service No. 21.
Real-Time Computer Science Corp. (RTCS)
1390 Flynn Rd.
Camarillo, CA 93010
(805) 987-9781

Syncra PC, from **Eastman Communications**, is a software package that allows error-free transfer of documents and files among IBM PCs and compatibles. The program must be operating on both the transmitting and receiving computers in order to transfer data. It can also communicate directly with corresponding Syncra software packages on DEC VAX and IBM OS/DOS mainframes and System/36 minis. Syncra PC uses an asynchronous protocol and can be transmitted between 1,200 and 9,600 bps. An automated feature allows completely unattended operation. A data-compression/compaction feature allows document and file sizes to be reduced by 50 percent or more. Retail price is $79. Reader Service No. 22.
Eastman Communications
1099 Jay St.
Rochester, NY 14650
(716) 464-5500

Boca Research's BOCARAM is available for IBM PCs, PC/XTs (including the XT 286), PC/ATs, and compatibles. BOCARAM fits into the XT 286 box, connecting to the 8-bit connector to expand its memory from 640K to 2 megabytes per board. The board conforms to EMS 3.2, which permits the use of application software packages that access memory up to 8 megabytes. BOCARAM software includes a RAM disk, print spooler, and memory diagnostic program in addition to the Boca Research Expanded Memory Manager driver. Prices range from $245 to $740, depending on the amount of memory. Reader Service No. 23.
Boca Research
6104 Congress Ave.
Boca Raton, FL 33431
(305) 997-6227

Discovery Systems has released an audio-cassette training program for Autodesk's AutoLISP, a training

course for AutoCAD users. The eight lessons provide a step-by-step program with complete instructions to create custom AutoLISP functions, custom menus, and other timesaving utilities. The price is $179. Reader Service No. 24.
Discovery Systems
34 Autumnleaf
Irvine, CA 92714
(714) 733-9890

American Computer & Peripheral has introduced an accelerator card that utilizes the 80386 microprocessor. The 386 TURBO board can bring a 6-MHz IBM PC/AT or compatible up to 12 MHz and an 8-MHz computer up to 16 MHz. Clock rates are switchable through software without rebooting. Software written for the AT (including DOS, ROM BIOS, EGA ROM, and so on) executes from a 1-megabyte cache memory. The board sells for $1,995. Reader Service No. 25.
American Computer & Peripheral Inc.
2720 Croddy Wy.
Santa Ana, CA 92704
(714) 545-2004

TEFT (Terminal Emulator and File Transfer) from **S. M. Vorkoetter Software** allows an IBM PC, PC/XT, PC/AT, PCjr, or compatible to be used as an intelligent terminal for communicating with a host computer or a BBS. Features of TEFT include VT100 terminal emulation, text file transfer, conversion of binary files to text files and vice versa, a batch mode, and baud rates from 50 to 9,600 baud (50 to 4,800 on a PCjr). The product requires 128K RAM, one disk drive, and an IBM-compatible serial adapter card. TEFT is not copy-protected and is priced at $60. Reader Service No. 26.
S. M. Vorkoetter Software
P.O. Box 872
Waterloo, Ont.
Canada N2J 4C3

SK DOS, from **Star-K Software Systems**, is a single-user operating system for 68xxx-based machines. This generic DOS is easily implemented on a new system and allows programs written for one system to run on many others. It includes more than 40 commands and system programs, including a 6809 emulator. SK DOS sells for $125. Reader Service No. 27.
Star-K Software Systems
P.O. Box 209
Mt. Kisco, NY 10549
(914) 241-0287

Electronic Specialists has released an RS-232 bus-protection device called Kleen Line. The Kleen Line security system is designed to suppress damaging line spikes caused by lightning or large electrical machinery. Units can be configured with any, or all, of the RS-232 bus lines protected. Model PDS-232 M/F, which guards lines 1, 2, 3, and 7, sells for $143. Reader Service No. 28.
Electronics Specialists Inc.
171 S. Main St.
Natick, MA 01760
(617) 655-1532.

Polytron Corp. has introduced PolyShell, a DOS extender and command interpreter that adds a Unix interface and much of the capability and flexibility of Unix to MS-DOS. The shell consists of two major components: the Command Interpreter, which can be used instead of or in conjunction with the MS-DOS command interpreter; and the PolyShell Utility Set, which includes several utilities previously associated only with the Unix operating system. PolyShell is invoked as a program under DOS, and any MS-DOS commands can be called from within the shell. PolyShell runs on the IBM PC, PC/XT, PC/AT, and compatibles with DOS 2.0 or later and requires at least 256K RAM. A hard disk is recommended. A single-user license costs $149. Reader Service No. 29.
Polytron Corp.
1815 N.W. 169th Pl.
Ste. 2110
Beaverton, OR 97006
(503) 645-1150

Graphics

Dynaware has released Dynaperspective, a 3-D solid modeling graphics software package for PC-DOS machines. Dynaperspective currently has drivers for the following graphics boards: EGA, Number Nine Revolution, Artist1, and Artist2. In addition to the graphics boards, Dynaperspective also supports most major plotters, printers, mice, and digitizing tablets. The package sells for $1,850. Reader Service No. 30.
Dynaware
1309 114th SE, Ste. 303
Bellevue, WA 98004
(206) 451-0200

Windows Draw from **Micrografx** is a free-form graphics program that runs under Microsoft Windows. Windows Draw images are object-based rather than pixel-based and thus achieve device independence, which allows the images to be printed with the maximum resolution of the printer rather than that of the computer. Windows Draw includes Windows ClipArt (a collection of Windows-compatible artistic images) and sells for $299. Reader Service No. 31.
Micrografx Inc.
1820 N. Greenville Ave.
Richardson, TX 75081
(214) 234-1789

Another graphics package designed for use with Microsoft Windows is Cricket Graph, from **Cricket Software**. Designed to run on the Macintosh, the package has page-layout capabilities and supports a variety of printers, plotters, and film recorders. It also offers a variety of editing and data manipulation capabilities: data can be sorted; grouped by ranges of values; smoothed; and transformed by logarithmic, trigonometric, exponential, and statistical functions. Cricket Graph sells for $295. Reader Service No. 32.
Cricket Software
3508 Market St., Ste. 206
Philadelphia, PA 19104
(800) 345-8112
(215) 387-7955

Surf3-D/Surf87 from **dogStar Software** is a 3-D surface plotting program written in Turbo Pascal and using TurboHALO graphics routines. The program can work with most MS-DOS display screens and printers. The program calculates and plots the surface of selected x,y functions or a function you supply and permits you

OF INTEREST

(continued from page 149)

to rotate the surface about any axis. You can select scaling factors, surface hatching, and other options. The 3-D plotting routines include optional 8087 math coprocessor support that is optimized for rotation. Source code is included. Surf3-D/Surf87 is a shareware product; a donation of $10 is suggested. Reader Service No. 33.

dogStar Software
P.O. Box 302
Bloomington, IN 47402
(812) 333-5616

The Hot Shot graphics printer interface for Commodore computers from **Omnitronix** supports graphics printing on most popular dot-matrix printers. It has a standard internal 1K × 4 graphics buffer that can be expanded to 8K to help speed up printing. Screen dumps can be set for reverse or inverse printing, and on many printers you can select enhanced, double-density printing of graphics screens and graphics characters. Hot Shot sells for $59.95. Reader Service No. 34.

Omnitronix Inc.
760 Harrison St.
Seattle, WA 98109
(206) 624-4985

For the Mac

The MacBus/RTI-800 series software, from **National Instruments**, allows control of the Analog Devices RTI-800 series analog and digital I/O boards with the Macintosh Plus using MacBus. The software enables users to program the boards with Microsoft BASIC, Megamax C, and LabVIEW (LabVIEW and C application examples are included and fully explained). MacBus/RTI-800 series software sells for $195. Reader Service No. 35.

National Instruments
12109 Technology Blvd.
Austin, TX 78727-6204
(800) 531-4742
In TX (800) IEEE-488
(512) 250-9119

MacroMind has realeased Maze Wars+, a real-time multiplayer game for the AppleTalk network that is a direct descendent of the classic Maze Wars game from MIT and Xerox PARC in the early 70s. A terminal program is built into the game to allow connection with another player via 1,200–2,400-baud modem or by direct null modem. Messages can also be passed back and forth. Maze Wars+ is not copy-protected and costs $49.95; site licenses are available for $15 per node. Reader Service No. 36.

MacroMind Inc.
1028 W. Wolfram St.
Chicago, IL 60657
(312) 871-0987

OWL International has released a hypertext system for the Mac called Guide. Guide incorporates many features of standard word processors and outline processors as well as additional facilities for information management, such as annotation and cross-referencing. Guide requires a 512K Mac, Mac Plus, or Mac XL and can work with any graphics program that supports the clipboard. It can be used with MacWrite, Microsoft Word, Aldus Pagemaker, or any program that can read MacWrite files. Guide sells for $134.95. Reader Service No. 37.

OWL International Inc.
14218 N.E. 21st St.
Bellevue, WA 98007
(206) 747-3203

Bering Industries has developed a line of 5¼-inch removable cartridge systems for the Mac. The new line of Bernoulli drives, called Totem, includes three models : a single 20-megabyte removable cartridge for $1,495; a dual 20-megabyte removable cartridge for $2,295; and a combination 20-megabyte removable cartridge plus a 20-megabyte fixed hard disk for $2,295. Bering also sells a 20-megabyte 5¼-inch fixed drive for the Mac for $795. Reader Service No. 38.

Bering Industries
250 Technology Circle
Scotts Valley, CA 95066
(408) 438-8779

For the PC

IOTools, from **Rhoads Software**, provides terminal-independent I/O mapping with a constant programming interface. The package gives you control over console, asynchronous, and parallel I/O as well as several useful library modules. It makes for easy management of the characters from the keyboard and includes more than 15 modules that export more than 200 procedures. Formats are available for MS-DOS/Logitech and Pecan, and the price is $79.50, or $950 for source code. Reader Service No. 39.

Rhoades Software
504 Meeting House Ln.
Kennett Square, PA 19348
(215) 388-2626

Command Plus, from **ESP Software Systems**, is a command processor for MS-DOS machines that simplifies and enhances MS-DOS' Command features as well as offering many additional features such as a batch programming language with a Pascal-like syntax called SCRIPT. Other new features include command macros, command recall, file browsing, a log facility, the ability to access environment variables from the command line, and the ability to select files using ESP's extended file-name pattern-matching facility. Command Plus sells for $79.95.Reader Service No. 40.

ESP Software Systems Inc.
11965 Venice Blvd., Ste. 309
Los Angeles, CA 90066
(213) 390-7408

Sapiens V8, from **Sapiens Software Corp.**, is a virtual memory manager for C programmers on the PC. Features include 8 megabytes of virtual memory workspace, the ability to link V8 libraries to C compilers, and software emulation of 64-bit architectures. Sapiens V8 sells for $300. Reader Service No. 41.

Sapiens Software Corp.
P.O. Box 7720
Santa Cruz, CA 95061-7720
(408) 458-1990

DDJ

ADVERTISER INDEX

Reader Service No.	Advertiser	Page No.
369	Aker Corporation	91
350	Aldebaran Laboratories	27
321	Alpha Computer Service	92
273	Alsys	49
396	AT&T Technology	1
250	Austin Code Works	113
182	BC Associates	123
159	Blaise Computing	2
217	Blaise Computing	129
263	Block Island Tech	59
161	Borland International	C4
384	Boston Software Works (The)	29
387	Bryte Computer	59
212	Burton Systems Software	74
*	Business Software	142
*	C Toolbox	138-139
181	C Users Group	74
122	Compu View	34
348	Creative Computer Software	58
*	Creative Programming	62
379	Crescent Software	86
268	Custom Software Systems	119
*	DDJ Back Issues	96
*	DDJ Listings	104
*	DDJ Subscriptions	36
203	Datalight	9
258	Desktop A.I.	78
127	Digitalk	13
89	Ecosoft, Inc.	143
138	Essential Software	C3
93	Fair-Com	115
373	Genesis Data Systems	80
*	Gimpel Software	83
*	Gimpel Software	151
291	Gold Hill Computers, Inc.	20-21
97	Greenleaf Software	131
351	Guidelines Software	73
132	Harvard Softworks	98
376	Hi-Tech Software	80
327	Integral Quality, Inc.	68
190	Intel Corporation	60
355	Jou Laboratories	94
294	Kurtzberg Computer Systems	97
101	Lattice, Inc.	145
118	Lifeboat	73
359	Lifeboat	108
257	Logitech, Inc.	69
135	Lugaru	85
*	M&T Catalog of Books & Software Tools	PB
313	Magma Systems	61
336	Magus Inc.	85
108	Manx Software Systems	7
317	Marshal Language Systems	56
285	MDS, Inc.	100
397	Meridian Software Systems	35
392	Metagraphics	100
95	MetaWare Incorporated	90
300	Micro Way	46
286	Microcompatibles	65
215	MicroHelp, Inc.	72
*	Micromint	67
105	Microprocessors Unlimited	74
380	Microsoft	4-5
*	Micro/Systems Journal	116
249	Mortice Kern Systems, Inc.	85
*	MS-DOS Tools & Enhancements	117
309	Nanosoft Associates	130
220	Nantucket Corporation	137
243	Norton Utilities (The)	100
251	Nostradamus	C2
227	Oakland Group, Inc.	81
254	Oasys	45
130	Orchid Technology	15
214	Periscope Co. Inc.	120
343	Phar Lap Software	65
239	PMI	84
283	Polytron Corporation	25
229	Port-A-Soft	99
129	Programmer's Connection	75-77
98	Programmer's Connection	33
133/141	Programmers Shop (The)	88-89
301-304	Programmers Shop (The)	93
*	Quaid Software	59
144	Quantum Computing	112
*	Raima Corporation	103
393	Ramnet	42
*	SAS Institute	55
168	Sapiens Software	82
391	Scantel Systems Limited LTD.	86
210	Scientific Endeavors	90
394	Secom Information Products Co.	95
114	Seidl Computer Engineering	66
85	Semi-Disk Systems	147
78	SLR Systems	78
40	Soft Craft Inc.	24
259	Softfocus	64
218	Software Directions, Inc.	57
314	Software Garden Inc.	73
347	Software Masters	140
170	Software Security, Inc.	71
372	Softway	64
142	Solution Systems	135
152	Solution Systems	126
236	Springer-Verlag	39
363	Summit Software Technology, Inc.	87
395	Sutrasoft	58
*	TeleOperating Systems	127
*	Texas Instruments	54
230	The Software Family	47
*	Turbo Pascal Tools	63
119	Turbo Tech Report	44
332	Unify Corporation	79
316	Upland Software	40
157	Vermont Creative Software	53
112	Wendin	11
282	Whitewater Group (The)	43
244	Workman & Associates	141
225	Xenosoft	99

*This advertiser prefers to be contacted directly: see ad for phone number.

ADVERTISING SALES OFFICES

Southeast /Midwest
Gary George (404) 897-1923

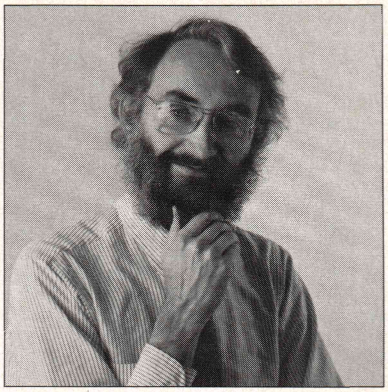
Northern California/Northwest
Lisa Boudreau (415) 366-3600

Northeast
Cynthia Zuck (718) 499-9333

Southern California/AZ/NM/TX
Michael Wiener (415) 366-3600

Director of Marketing and Advertising
Ferris Ferdon (415) 366-3600

SWAINE'S FLAMES



Steve Jasik could reasonably call his *MacNosy V2 Documentation V2.50* the most eagerly awaited manual of the year. It's not that drooling hordes of programmers have been demanding this Macintosh disassembler—it's still an underappreciated marvel—but those of us who grappled with the original documentation were wet-chinned with anticipation.

The new manual fairly describes Nosy as an information-recovery tool for the Mac, a description justified by the on-line Inside Mac feature alone. I was so eager to use Nosy that I rashly agreed to help revise the original manual, bailing out only when I realized that I would have to understand Nosy intimately to improve the documentation and that there was no way I was going to get Nosy literate from Steve's manual.

Others were more successful, and the resulting documentation is not only comprehensible but also strewn with uncloneable Jasikisms from the Head Nose.

Nosy would be a handy tool for, say, an independent programmer developing desk accessories that worked intimately with Microsoft Excel. Tools for programmers working in teams have been slower in coming than such individual tools, but computer-aided software engineering (CASE) has taken a step forward with the arrival of commercially priced 32-bit machines at a time when successes in CAD/CAM are ripe for translation to general software development.

Rich Carpenter of Index Technologies, a Cambridge, Massachusetts, CASE company, argued at the Personal Computer Forum in Phoenix this spring that current software-development approaches have been borrowed from engineering situations in which, for example, it was prohibitively expensive to move elevators after putting up inside walls. Software development needs its own models for, and tools in support of, design, specification, prototyping, version management, testing, debugging, and documentation.

Software development also needs better tools. One of the advantages of C is that you can get fairly efficient code out of primitive compilers, which could be a fair assessment of 1986-vintage microcomputer compiler technology. Not-so-primitive optimizing compilers are now arriving: I know of five optimizing C compilers released or in the works, and Microsoft's FORTRAN compiler is one of the most sophisticated optimizing compilers on personal computers. It will, for example, turn

```
y=sin(x)**2
```

into the faster

```
temp=sin(x)
y=(temp*temp)
```

Scientific and engineering users of personal computers can expect more than just a fast FORTRAN from software companies, though. Both Lotus and Borland now have engineering and scientific divisions dedicated to developing products for this group, which Lotus says accounts for 17 percent of its existing user base. Lotus is even considering starting a magazine for scientists and engineers.

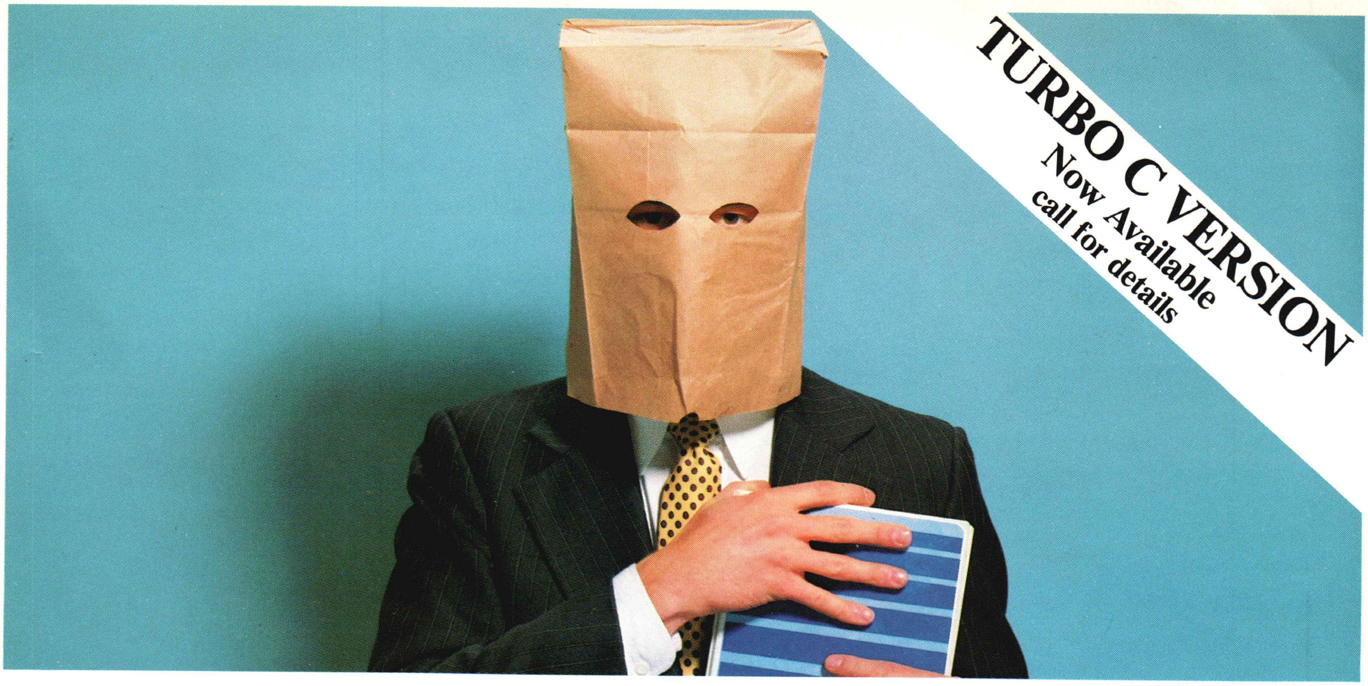
Legal issues: ADAPSO's efforts to get software vendors to voluntarily provide substantive warranties rather than "as is" disclaimers on products has led a member of the California Assembly, Gloria Molina, to recommend against legislation to force warranties. The Copyright Office has judged Lotus' 1-2-3 user interface uncopyrightable from a look-and-feel standpoint because it is basically text. And several senators are reintroducing a bill to create an Information Age Commission, whose purpose would be to study the impacts of high technology.

Memorable moments at the Personal Computer Forum: hearing that the personal computer industry looks willing to give Bill Gates a billion-dollar company in exchange for a little stability; Mitch Kapor deftly fending off an embarrassing question about the Lotus look-and-feel suit by attacking David Bunnell's editorial on the Georgia sodomy law; and the following story:
"The editor kept calling our product a database and I said, 'It's really a programming language.' But he said, 'We don't cover languages, Bruce.' So I said, 'Well, really it's more of a database.' Then everything was fine."

This story was related by a software company president during a dinner at which several software company CEOs told M&T executives horror stories about the technical incompetence of the computer press, including the story of the product that got rave reviews for a "feature" that was really a bug the company had been trying to eliminate for months. The hard work that goes into developing a good program deserves competent reporting.

Michael Swaine

Michael Swaine
editor-in-chief

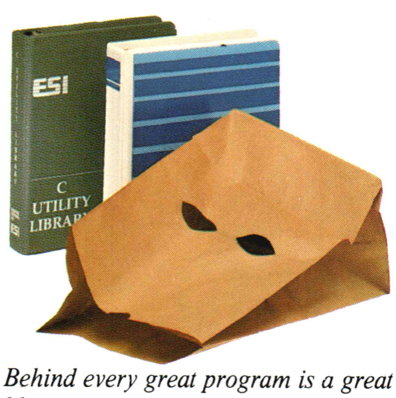


Some Of The Most Famous Faces In Software Use Our C Functions

Our famous customers are a little camera shy. It's not that they are embarrassed by being Essential C Utility Library users. They just don't want us shouting their names from the roof tops.

The prestige of our users is not the primary reason to buy the Essential C Utility Library. The increased speed, features, and size efficiency of our products are the factors that demand their use. Our library contains *over 400 functions,* all designed with elegance in mind.

However, should curiosity get the best of you, call us at 201-762-6965 and we'll drop a few highly impressive names on you.



Behind every great program is a great library.

What's a Library Without a Librarian?

Our library comes complete with a sophisticated source code librarian. Now you can maintain current versions and conserve disk space. We want your development work to go as smoothly as possible.

No Royalties, 30 Day Guarantee

If within 30 days you don't find our library totally satisfactory, bag the whole thing and receive a complete refund. There are no royalties associated with the library.

Functions At A Glance

- Fastest screen output available.
- Save/Restore color screens in 1/10 sec.
- Pop-up block cursor menus
- Save/Restore windows to disk or memory
- 50 functions for business graphics
- dozens of string formats
- time and date arithmetic
- julian and day-of-week
- Ctrl-Break key trapping
- Field oriented data entry
- Stuff keyboard buffer
- 18 Mouse control functions
- Execute programs and batch files
- Disk error trapping
- Determine space available
- 40 functions to process characters and words
- Insert, delete, extract, index, translate
- Tested, easy-to-follow examples
- Demo programs with source code
- All source code included

Documentation:
Thorough, comprehensive, 260 pages

Compatible C Compilers:
Microsoft, Lattice, Computer Innovations, Aztec, Mark Williams, DeSmet, and Wizard

$250.00

Do Your Homework

The library you buy can influence the rest of your programming life. We encourage you to do some checking before making a decision. When you've done your homework, you'll choose Essential. Call our support staff of experienced C programmers and find out before you buy how things will be after your check clears.

To order or for support call: 201-762-6965

For foreign orders contact:
England: Gray Matter Tel. (0364) 53499
Japan: Lifeboat Inc. of Japan Tel: 293 4711
West Germany: Omnitex Tel. 07623-61820

Essential Software, Inc.
P.O. Box 1003, Maplewood, New Jersey 07040

Circle **no. 138** on reader service card.

NEW Turbo C®

Turbo C: The fastest, most efficient and easy-to-use C compiler at any price

Compilation speed is more than 7000 lines a minute, which makes anything less than Turbo C an exercise in slow motion. Expect what only Borland delivers: Quality, Speed, Power and Price.

Turbo C: The C compiler for amateurs and professionals

If you're just beginning and you've "kinda wanted to learn C," now's your chance to do it the easy way. Like Turbo Pascal, Turbo C's got everything to get you going.

If you're already programming in C, switching to Turbo C will considerably increase your productivity and help make your programs both smaller and faster. Actually, writing in Turbo C is a highly productive and effective method—and we speak from experience. Eureka: The Solver™ and our new generation of software have been developed using Turbo C.

Turbo C: a complete interactive development environment

Free MicroCalc spreadsheet with source code

Like Turbo Pascal® and Turbo Prolog,™ Turbo C comes with an interactive editor that will show you syntax errors right in your source code. Developing, debugging, and running a Turbo C program is a snap.

Turbo C: The C compiler everybody's been waiting for. Everybody but the competition

Borland's "Quality, Speed, Power and Price" commitment isn't idle corporate chatter. The $99.95 price tag on Turbo C isn't a "typo," it's real. So if you'd like to learn C in a hurry, pick up the phone. If you're already using C, switch to Turbo C and see the difference for yourself.

System requirements

IBM PC, XT, AT or true compatibles. PC-DOS (MS-DOS) 2.0 or later. One floppy drive. 320K.

*Introductory price—good through July 1, 1987

Technical Specifications

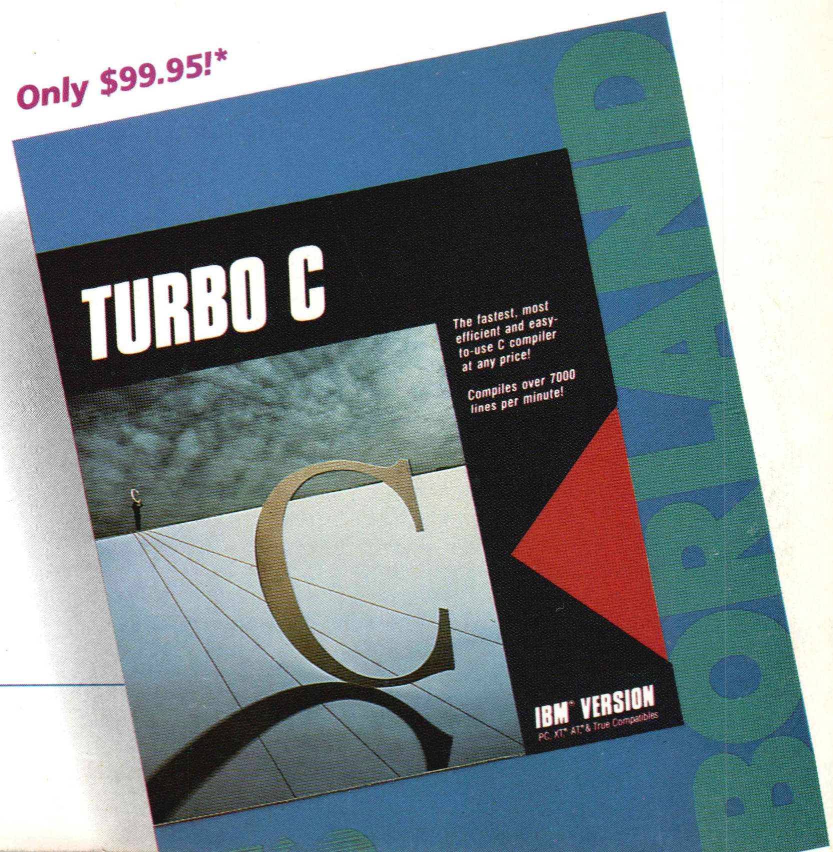
- Compiler: One-pass compiler generating linkable object modules and inline assembler. Included is Borland's high performance "Turbo Linker." The object module is compatible with the PC-DOS linker. Supports tiny, small, compact, medium, large, and huge memory model libraries. Can mix models with near and far pointers. Includes floating point emulator (utilizes 8087/80287 if installed).
- Interactive Editor: The system includes a powerful, interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code.
- Development Environment: A powerful "Make" is included so that managing Turbo C program development is highly efficient. Also includes pull-down menus and windows.
- Links with relocatable object modules created using Borland's Turbo Prolog into a single program.
- ANSI C compatible.
- Start-up routine source code included.
- Both command line and integrated environment versions included.

Sieve benchmark (25 iterations)

	Turbo C	Microsoft® C	Lattice C
Compile time	**3.89**	16.37	13.90
Compile and link time	**9.94**	29.06	27.79
Execution time	**5.77**	9.51	13.79
Object code size	**274**	297	301
Price	**$99.95**	$450.00	$500.00

Benchmark run on a 6 Mhz IBM AT using Turbo C version 1.0 and the Turbo Linker version 1.0; Microsoft C version 4.0 and the MS overlay linker version 3.51; Lattice C version 3.1 and the MS object linker version 3.05.

Only $99.95!*



BORLAND
INTERNATIONAL

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

Vive la différence

For the dealer nearest you or to order by phone call

(800)255-8008

in CA (800) 742-1133 in Canada (800) 237-1136

CIRCLE 161 ON READER SERVICE CARD